linux

目录权限

rwx对于目录，内容是文件名。

r 读到目录的文件名

w 修改文件名(文件或目录),建立新的文件与目录，移动或更名

x cd 能进入到目录里面

20200420

监控主机看要被监控主机的地域来开设机房的机子，传输稳定减少变数

被监控机子要提供验收范围(port与服务)

20200425

mysql all数据库备份与还原

备份

mysqldump -uroot -pabc123 --all-databases > all.sql

还原

mysql -uroot -pabc123 < all.sql

bash

\rm -r 目录 #\忽略掉alias的指定选项

cp 链接文件 文件 #命令指行后的文件不是链路接文件而是被链路接指向的源文件

cp -d 链接文件 文件 # 得出的文件也是链路接文件

cp -a #将文件的所有属性一起复制

20200427

数据库用户，如果网站不是太重要，或是内部服务器，数据库的用户可以是 root

20200429

mysqldump -uroot -p -B database > database.sql

#使用-B的好处就是导出的数据库文件中已经存在创建库和使用库的语句，不需要手动在要还原的机子上创建库的操作，可直接还原。

还原

mysq -u root -p '123456' < database.sql #还原时不需要指定恢复的数据库

linux

查找etc目录下容量大于50K且小于60K的文件并ls -l查看详细信息

find /etc -size +50k -a -size -60k -exec ls -l {} \;

查找etc目录下容量大于1500K以及容量等于0的文件

find /etc -size +1500k -o -size 0 -exec ls -l {} \;

# -a 且 , -o 或

查看文件的隐藏属性

lsattr file

设置隐藏属性

chattr [+aiS]

locate 文件 #locate通过数据库/var/lib/mlocate查找文件,不用找硬盘速度快

20200430

SUID

文件具有suid权限时，代表当用户执行此二进制文件时会暂时具有文件拥有者的权限

SGID

目录具有sgid时,代表用户在此目录下新建的文件的用户组会与该目录的组名相同

SBIT

目录具有sbit时，代表该目录下用户建立的文件只有自己与root能删除

设置

SUID 4

SGID 2

SBIT 1

如 4775 #4为设置suid

7775 #设置 suid,sgid,sbit

计算机安全，通信

截获

被动攻击，如cain软件利用ARP欺骗把自身的mac地址(假网关地址)发给同网段各主机,这样同网段各主机发送报文都能截获，如user/pass

篡改

主动攻击,如cain软件利用arp欺骗把自身的mac地址(假网关地址)发给网段，各主机访问外网时cain提供假域名给各主机,cain主要去访问真域名，再把信息转给各主机，这是一种DNS劫持。

中断

后端主机控制大量肉鸡发送大量无用的数据包给服务器，占用下载带宽，使其正常服务不能使用.DDOS攻击

伪造

同一网段ip邦定mac来上网，若这一台机子关机，另一台机子可以有机会使用相网的ip上网，达到伪造的效果

磁盘/硬盘

磁片

主轴马达

机械手臂与磁头

文件系统

通常是一个分区对应一个文件系统，现在多数是一个挂载点对应一个文件系统

一个文件有文件实际内容与文件的权限属性等相关的参数

inode

记录文件的权限与属性相关的参数，一个文件站用一个inode，同时记录该文件的实际数据所在的区块号码，有编号

数据区块

实际记录文件的内容，若文件太大会占用多个区块，有编号

超级区块

记录文件系统的整体信息，包括Inode与数据区块的总容量，使用量，剩余量，以及文件系统的格式与相关信息

索引式文件系统

通过Indode可以一口气读取所有的数据区块,读写性能较好

FAT(u盘类)是非索引式文件系统，没有inode,每个区块的号码都记录在前一个区块中,不能一次性读取。所谓的碎片整理就是把太过于离散的区块整理集中。

ext2文件系统

文件系统一开始就会将inode和数据区块规划好，除非重新格式化，否则inode和数据区块是不会变的

如果文件系统高达数百G，将所有的inode和数据区块放在一起不易管理，所以会有多个区块组(block group)

文件系统最前面有一个启动扇区(boot sector),可以安装引导程序，我们可以安装不同的启动引导到别的的文件系统的的boot sector，这样就不用覆盖磁盘的唯一MBR，实现多引导

区块组

每个区块组都有独立的超级区块，Inode,数据区块

数据区块

inode table

inode,记录区块号码可以有12个直接指向区块的号码，

一个间接

如果文件太,再拿一个区块来当作记录区块号码的记录区

一个双间接，一个三间接

inode对应表

记录空白的与使用的In(空区块)ode,哪些可以使用哪些不可以使用

区块对应表

记录可使用的区块(空区块)，不可使用的区块

文件系统描述

超级区块

事实上除了第一扇区会有超级区块外，后续的区块组不一定含有超级区块。若含有则是主要为第一个超级区块做备份

目录

linux下新建一个目录时，会分配一个Inode与至少一块区块给目录，inode记录目录相关的权限与属性，并可记录分配到哪块区块号码，而区块则是记录目录下的文件名与该文件名占用的Inode号码数据

文件

新建一个文件时，会分配一个inode与相对该文件大小的区块数量给文件

/

由于目录树是由根开始，因此系统通过挂载的信息可以找到挂载点的Inode号码，就能找到根目录的inode内容，再通过该inode读取根目录区块内的文件名数据，再一层一层往下读到正确的文件名

日志文件系统

inode对照表，数据块对照表，超级区块经常变动，称为元数据

inode tabes，数据块区块称为数据存放区域

万一如果出现停电或发生错误导致系统中断，数据存放区域与元数据产生不一致的情况，那么系统重新启动时就要通过超级区块和文件系统状态来进行数据一致性检查，浪费大量时间，所以就有日志文件系统的出现。

日志文件系统，系统中规划了一个区块专门记录文件写入与修改的步聚，当数据出现不一致性时，只需检查该区块即可，不用针对整个文件系统检查。

异步处理

cpu处理文件时，文件要先放到内存中，如果编辑文件太大，因为内存比磁盘速度快，所以等待读写入磁盘的时间太久，没效率。

异步处理就是当系统加载一个文件到内存后，如果该文件没有被修改，则内存区段的数据就设置为clean,若修改则设置为dirty,此时操作都还在内存中进行，并没有写入磁盘。系统会不定时的把内存中dirty的数据写入磁盘，保持数据一致性。也可以手动sync命令

系统会将常用的数据放置到内存缓冲，加快文件系统的读写操作，所以Linux的物理内存都会被用光，正常现象，加速系统性能。

20200501

linux的内核通过VFS来管理读取文件系统

xfs

centos7.x默认的文件系统

数据三个部分

数据区(data section)

与ext2的区块群组一样，不同的是xfs的inode与区块都是系统需要用到时动态配置产生，所以格式化操作超快

文件系统登录区

有点日志区块，还可以指定外部磁盘来作为xfs文件系统的日志区块

实时运行区

当有文件新建立时，xfs会在这个区段里面找一个到数个的extent区，将文件放置在这个区块内，等分配完毕后再写入到data section的inodae与区块中

20200504

硬链接

不能跨文件系统

不能跨接目录

如果链接的比如是etc目录,那目录下的所有文件都要硬链路接，工作环境太过复杂

硬链接就是多个文件名指向同一个Inode，文件系统的inode与数据区块一般是不会变化的

创建硬链接

ln /etc/crontab . #创建硬链接到当前目录

[root@localhost ~]# ls -il crontab /etc/crontab

4372557 -rw-r--r--. 2 root root 451 Jun 10 2014 crontab

4372557 -rw-r--r--. 2 root root 451 Jun 10 2014 /etc/crontab

第3个字段2，代表是有多少个文件名链路接到这个Inode

软链接（快捷方程式)

符号链接就是建立一个独立的文件，而这个文件会让数据的读取指向它链接的那个文件的文件名

创建软链

ln -s /etc/crontab crontab2

[root@localhost ~]# ll -i crontab2

8414548 lrwxrwxrwx. 1 root root 12 Apr 30 00:48 crontab2 -> /etc/crontab

当源文件被删除后，符号链接文件会打不开，实际上是找不到原始文件名而已

符号链接所建立的文件为一个独立的新文件，所以会占用inode与区块

目录链接数

当我们建立一个新目录时，新的目录链接数为2，上层目录的链接数增加1

磁盘分区

lsblk #列出系统上的所有磁盘列表

blkid #列出设备的uuid参数， uuid全局唯一标识符，linux会将系统内所有的设备给于一个标识符，标识符可以拿来挂载或使用

parted /dev/sda print #列出磁盘的分区表类型与分区信息

gdisk/fdisk /dev/sdb #对/dev/sdb磁盘进行分区管理(增删等)，q退出不保存，注意不要去处理一个正在使用的分区

一般先用lsblk找到磁盘，然后用parted找出分区格式(gpt or MBR), 再利用fdisk分区(MBR格式的话)

用fdisk分好区并w保存后，分区表并没有立即更新，需要

partprobe -s #更新linux内核的分区信息

查看内核分区的信息

lsblk

cat /proc/partitions

磁盘格式化(创建文件系统)

mkfs.xfs /dev/sdb6 #一般使用默认参数就行，除非有其它额外的要求，如-f, 强制格式化,如果原来已有文件系统

mkfs.ext4

文件系统检验

xfs\_repair /dev/sdb4 #恢复命令,万一服务器停电或硬件软件出错导致文件系统发生错乱

通常文件系统出问题时才使用(单人模式下)，正常状况使用此命令可能会造成系统损坏，且被检验的硬盘分区不能挂载，要在卸载状态下使用

20200505

挂载与卸载

单一文件系统对应单一目录(挂载点)，目录应该是空目录才行，如果不是那原先存在目录下的文件在该目录被挂截之后会暂时先隐藏起来(并不是消失)，当目卸载之后才会看到

mount #什么都不接就是列出目前挂载信息

mount -a #依照配置/etc/fstab的数据把所有没挂载的磁盘都挂载上来

mount -t #指定挂载的文件系统种类。ceoto 7 会自动分析文件系统种类来尝试挂载设备(根据文件系统的超级区块与linux自己的驱动程序去测试)，测试成功则自动挂载.

主要参考这两个文件来

/etc/filesystems 系统指定的测试挂载文件系统类型的优先级

/proc/filesystems linux系统已加载的文件系统类型

如何知道系统有没有相关的文件系统的驱动程序呢

/lib/modules/$(uname -r)/kernel/fs/

mount 文件系统 挂载点

如 mount /dev/sdb6 /newfile

mount UUID /newfile

挂载cd/dvd, u盘(vfat格式)

先blkid查看设备名称,再mount挂载,一般cd/dvd挂载后都是只能只读，使用率为100%，u盘最后是vfat格式,linux本身不支持ntfs格式，如果有中文文件名的话，可以指定挂载参数语系

重新挂载或挂载不特定目录

如当/为只读状态，可以重启系统或重新挂载

mount -o remount,rw,auto /

可以将某个目录挂载到另一个目录，有点类似软链，特殊时才会用

mount /var /test/var #将/var这个目录暂时挂载到/test/var下面

卸载

umount 文件系统设备名/挂载点

umount /dev/sdb6

umount /test

磁盘/文件系统自定义参数

mknod #设置如/dev/sda 文件名，一般系统会自动帮设置好

xfs\_admin #设置lable name与uuid

20200507

启动挂载

/etc/fstab #开机就启动的文件挂载文件，注意 / 必须是要先挂载的且必须挂载

六字段意义

[设备/uuid] [挂载点] [文件系统类型如xfs] [文件系统参数] [dump] [fsck为]

设备/uuid

可以是设备名如/dev/sda2,也可以是uuid,也可以是label名称

挂载点

一般是目录

文件系统类型

xfs, ext4，nfs等

文件系统参数

rw,auto,exec等相关参数

dump

备份方案太多一般不用,为0即可

fsck

检验扇区,早期的系统使用,xfs不使用,xfs会自己检验,为0即可

/etc/fstab修改好后，一定要测试语法正确性, mount -a 检验成功与否

/etc/fstab是启动配置文件，不过实际挂载信息是记录到/etc/mtab与/proc/mounts这两个文件，万一如果/etc/fstab写入的数据出错导致无法顺利启动而进入单人维护模式,就得重新挂载 /, 此时/为只读mtab这两个文件不能写。

mount -n -o remount,rw /

特殊设备loop挂载(镜像文件不刻录就挂载使用)

cd/dvd镜像文件

mkdir -p /data/centos\_dvd

mount -o loop /tmp/Centos-7.0.XXXDVD.iso /data/ceotos\_dvd #这就就能够不需要将iso刻录成dvd或光盘就读取到内部的数据了，并且可以修改里面的文件，这就了为什么镜像文件会提供md5验证码给用户确认该镜像没有问题

建立大文件以制作loop设备文件

比如一开始分区不合理，只有/目录有多余的容量，可以在/里面制作出一个大文件用来挂载，就相当于多了一个分区

dd if=/dev/zero of=/srv/loopdev bs=1M count=512

#if input file,输入文件,/dev/zero是会一直输出0的装备

#of outputfile 将一堆0写入到后面的文件中，

block 相当于文件系统的区块

count 总共有多少bs

mkfs.xfs -f /srv/loopdev

mount -o loop /srv/loopdev /data/file

交换分区

交换分区就是如果内存不足时，会把内存中不常用的数据或程序暂时放到磁盘上的交换分区里，以空出内存给后续程序或数据使用

两种创建交换分区的方式

使用磁盘分区来格式化一个swap文件系统

先用分区出来一块如1G的分区

mkswap /dev/sdb6 #格式他生成一个是swap文件系统

swapon /devsdb6 #启动交换分区

swapon -s #查看交换分区或使用free 查看

开机启动vi /etc/fstab

/dev/sdb6 swap swap defaults 0 0 #第二栏没有挂载点

使用大文件来格式化生成一个swap文件系统

dd if=/dev/sero of=/srv/swap\_dev bs=1M count=1024 #生成大文件

mkswap /srv/swap\_dev #格式化swap文件

swapon /srv/swap\_dev #启动

开机启动vi /etc/fstab

/srv/swap\_dev swap swap defaults 0 0 # 第一栏必须是设备名不能是uuid,因为系统只会查询设备的uuid

关闭swap

swapoff /dev/sdb6

swapoff /srv/swap\_dev

ls -l 显示的total数值

ll -sh,目录下的total值是文件数据区块数量\*区块大小值

root@localhost ~]# ll -sh

total 10M

4.0K -rw-r--r--. 1 root root 120 Apr 19 11:23 1.txt.tar.gz

4.0K -rw-------. 1 root root 1.3K Aug 11 2019 anaconda-ks.cfg

4.0K -rw-r--r--. 2 root root 451 Jun 10 2014 crontab

crontab的实际大小是451bytes，但这个文件占用一个区块(4k大小)

20200509

文件压缩

压缩技技原理

如数字1用1个字节表示,事实计算机的最小计算单位是1bit，所以数字1就是00000001,利用一些特殊的算法通过把前7个0"丢掉“就达到了节省空间的目的

如1111111111111111111111110，前面有十几个1甚至更多,可以用15\*1来表示存储之类达到节省空间

gzip

替代compress,最常用的压缩命令,gzip压缩后的文件可以被windows的winrar和7.zip解压

gzip 1.txt # 会自动生成1.txt.gz，.gz后缀名文件,注意源文件1.txt会不再存在

gzip -d 1.txt.gz #解压缩得到1.txt, gz文件会不再存在

gzip -v 1.txt #压缩的同时显示压缩比

zcat/zmore/zless/zgrep 可以用来查看或查找被压缩的文件

gzip -9 -c service > service.gz #-c将原本要压缩的内容变成文字类型从屏幕输出，然后利用>输出到service.gz(手动建立)，这样原文件与压缩文件都同时存在

bzip2 与gzip用法一样，压缩比更好，后缀名bz2

xz用法与gzip用法一样，压缩比更好，后缀名 xz

xz -l abc.xz #显示压缩前后容量对比

xz -k abc.txt #压宿文件同时保留源文件

tar

gzip等压缩软件只能对单一文件解压缩,tar是将多个文件或目录进行打包并能结合gzip解压缩命令

tar [-jzJ]cv -f 打包的文件名 要打包的文件 #打包并压缩

tar [-jzJ]tv -f 压缩的文件 #查看文件

tar [-jzJ]xv -f 压缩的文件 [-C 目录] #解压文件

-p #保留备份数据的原始权限与属性

-P #保留绝对路径,即允许备份数据中含有根目录,慎用 #大p

--exclude=文件 #不包含某个文件

备份/etc

tar -zpcvf /root/etc.tar.gz /etc

查看tar内部文件

tar -ztvf /root/etc.tar.gz

解压.tar.gz 到/tmp目录下

tar -zxvf /root/etc.tar.gz -C /tmp

打包某目录但不包含特定目录

tar -zcvf /root/system.tar.gz --exclude=/root/etc\* \

--exclude=/root/system.tar.gz /etc /root #打包etc和root目录但不包含root目录下的etc相关文件与自己本身,exclude最好放在tar.gz后面

备份比某个时刻要新的文件

[root@localhost ~]# ll /etc/passwd

-rw-r--r--. 1 root root 1498 Apr 29 05:09 /etc/passwd #日期为20200429

tar -zcvf /root/etc.newer.then.passwd.tar.gz --newer-mtime="20200429" /etc/\* #打包比20200429要新的mtime的文件

tar -ztvf /root/etc.newer.than.passwd.tar.gz | grep -v "/$" # 调用grep找出非/结尾的文件就是我们要的

tar -cvf file.tar # 仅是打包文件称为tarfile

tar -zcvf file.tar.gz #有压缩的支持称为tarball

tar可以将文件打包到某些特定的设备中，如磁带tab,磁带是一次性读取/写入设备，不能用cp来复制,如将/home /root /etc备份到磁带/dev/st0

tar -cvf /dev/st0 /home /root /etc

xfs文件系统的备份与恢复

xfsdump #备份，可以完整备份，增量备份，注意只能备份已挂载的文件系经

xfsrestore #还原，可以还原守整备份，增量备份(要安顺序level0->level1---),可以还原指定的文件, -i交互模式

光盘写入工具

先将所需要备份的数据创建成为一个镜像文件iso

将该镜像文件刻录到cd或dvd中

一般用图形界面软件来操作即可，无需用命令行工具

20200511

dd

制做文件或备份功能(基本直接读取扇区)

dd if="input file" of="output\_file" bs="block\_size" count="number"

if 就是Inputfile ,可以是设备

of 主是outputfile, 可以是设备

bs 设置一个block大小默认512byte(扇区大小)

count 多少个bs

复制/etc/passwd到/tmp/passwd.back中

dd if=/etc/passwd of=/tmp/passwd.back #

dd if=/dev/sr0 of=/tmp/system.iso #复制刻录好的光盘的内容备份成为镜像文件

把上面的iso刻录到U盘，假设/dev/sda为u盘

dd if=/tmp/system.iso of=/dev/sda #可以linux镜像文件写样做u盘就具有可启动功能了，然后可以安装Linux系统

将/boot文件系统通过dd备份下来，假设/boot挂载设备为/dev/vda2

dd if=/dev/vda2 of=/tmp/vda2.img #文件大小与磁盘大小一样，哪怕磁盘只使用50%

将/dev/vda2完整地复制到另一个硬盘分区上，假设我们已经分区好一个比vda2大的分区sda1

不需要格式化sda1

dd if=/dev/vda2 of=/dev/sda1

xfs\_repair -L /dev/sda1 #先清楚一堆log

#] uuidgen

948gjei-egeg-48g05656-56

xfs\_admin -U 948gjei-egeg-48g05656-56 /dev/sda1 #这两行是用于创建新的uuid，因为dd复制时连同uuid都复制过来了

mount /dev/sda1 /mnt #挂载发现/mnt与/boot一模一样

xfs\_growfs /mnt #系统放大空间

dd是将原本旧的硬盘分区上面的扇区数据整个复制过来，连同超级区块，启动扇区元数据等，所以不用格式化

如果想创建两块一模一样的磁盘，只要执行类似 dd if=/dev/sda of=/dev/sdb, sdb不用分区与格式化，因为命令包括MBR与分区表都复制过sdb了

cpio

可以备份任何东西，包括设备文件，要结合find与管道 |

也可以将系统数据完整的备份到磁带

备/boot下的所有文件到/tmp

cd /

find boot | cpio -ocvB > /tmp/boot.cpio #需要去掉根目录，与tar 的-P一个道理

20200512

vim

一般命令模式

ctrl + f 向下移动一页

ctrl + b 向上移动一页

0或Home键，移动到一行的最前面的字符处

$ 或END， 移动到一行的最后面的字符处

G移动到文件的最后一行

gg移动到文件的第一行

/word 向光标之下寻找一个名为word的字符串

?word 向光标之上寻找word

n 重复前一个查找的操作,如/word继续查找

N 与n相反，反向查找

:n1,n2s/word1/word2/g 替换n1到n2行的word1变成word2

:1,$s/word1/word2/g 从第一行到最后一行全都替换在word2

:1,$s/word1/word2/gc 从第一行到最后一行都替换，但替换前让客户确认

x, X x向后删除一个字符, X向前删除一个字符

dd 删除光标所在的那一行

ndd n为数字,向下删除n行

yy 复制光标所在那一行

nyy 复制光标以下的n行

p与P p为将已复制的内容在光标下一行粘巾,P为上一行

u 恢复前一个 ctrf + r 重做上一个操作

. 重复前一个操作

一般模式切换到编辑模式

i与I， i在光标处插入, I在本行的第一个非空字符处插入

a与A a在光标下一个字符出插入, A在光标所在行的最后一个字符处插入

o与O o在光标所在的下一行插入新的一行,O在光标所在处的上一行插入新的一行

r和R r替光标所在的字符一次, R一直替换直到esc

编辑模式

命令行模式

:w 保存

:w! 强制保存

:q 退出

:q! 强制退出

:wq 退出后保存

:w filename 将编辑的数据保存为另一个文件

:! command 暂时退出vim执行下命令

:set nu 显示行号

:set nonu 取消行号

vim缓存，恢复与打开警告

当修改还没来得及保存退出时，再重新vim时会出现警告,缓存文件为.file.swp。或多人打开并编辑了file

如果是其他人在编辑，可以选 择O模式只读

如果是未来得及保存则可以

选R，使用swp来恢复文件再决定要不要保存，这样可以救回来之前没保存的修改，退出后记得删除file.swp

或者不要恢复修改直接删除.swp文件

可视区块

Ctrl + v 选中之后可以d删除或y复制后再p粘贴

多文件

多窗口

在用vim打开一个文件后，在命令行输入

:sp filename , 打开一个新窗口，如果有file表示打开一个新文件，没有file则打开同一个文件

ctrl + w + 下 先按住ctrl再按w,然后放开所有的健, 再下 下 则光标移动到下方的窗口

ctrl + w + 上 移动到上方的窗口

:q 退出窗口

命令补全

20200514

利用iconv可以进行文件编码的转换

dos2unix, unix2dos可以变更文件的每一行的换行符,如windows与linux文件的换行符不一样

dos(windows)文件的换行符为^M$,称为CRLF

unix换行符为 $, 区别少了^M

bash

控制计算机硬件的是操作系统的内核,我们用过shell输入命令与内核沟通，让内核来操作硬件准确工作，bash属于壳程序(shell),其它应用程序也称为壳程序

type 命令可以查看命令是外部命令还是bash内置的命令,后接执行命令，不能是文件

[root@localhost ~]# type cd

cd is a shell builtin

# cp /test/1.txt /12/2.txt /3/3.txt \

>/5/5/txt /root #复制4个文件到/root目录下, \转义回车键，\后面要紧接着ENTER,不能多空格,因为\仅转义下一个字符

提示符命令行下

ctrl + u 或 ctrl + k 从光标处向前删除命令或向后删除命令

ctrl + a 或 ctrl +e 从光标处移动到整个命令的最前面或最后面

一般登录后,linux会根据/etc/passwd设置一个shell，默认bash

环境变量通常以大写字符表示

变量，类型默认字符串

变量的设置，变量与变量内容以一个=号来连接

name=VBird

等号两边不能接空格,变量内容有空格需用双引号或单引号结合起来

name = VBird name=Vbird good #错误

变量只能是英文与数字且不能数字开头

2myname=VBird #错误

双引号内的特殊字符可以保持原本的属性

var="lang is $LANG" -> echo $var -> lang is zh\_CN.UTF8

单引号内的特殊字符为一般字符

var=‘lang is $LANG' -> echo $var -> lang is $LANG

可用\转义符将特殊字符([ENter], \ , ', 空格等)变成一般字符

name=Vbird\'s\ good -> echo $name -> Vbird's good

在一串命令中需要借由其它命令提供信息时,可以使用反单引号`或$()

version=$(uname -r)

变量扩增内容可以用${}字符

PATH=${PATH}:/home/hello

若变量需要在子进程中执行，

export 变量 #使其在为环境变量

取消变量

unset 变量

设置变量代替工作目录

work="/home/good" #不用""也行

env 显示所有的环境变量

RANDOM 生成随机数的变量, 0~32767

set 观察显示所有的环境变量与用户自定义变量

PS1='[\u@\h \W]\$ ' 命令提示符就是 [root@www~]#

$ shell的PID

echo $$ 2787

? 上个执行命令的返回值

ls -h -> echo $? -> 0 #命令能成功执行就返回0，否则就返回非0

export 自定变量转成环境变量

export myname #使myname变成环境变量，这样myname就可以在子进程中使用，export后没接变量时，会列出所有的环境变量

父进程与子进程

Linux登录之后得到一个BASH(父进程),在这Bash下面所执行的任何命令(子进程)都是由这个BASH所衍生出来的.如在父进程的bash下面执行bash进入到另一个界面(子进程)，原本的Bash就会暂时sleep，当子进程bash exit退出后才回到原本的BASH。

子进程仅继承父进程的环境变量,不会继承自定义变量

declare 将环境变量转成自定义变量

为什么环境变量的数据可以被字进程引用?

因为内存配置关系，启动一个shell时，操作系统会给这个SHELL在内存中分配一个区域，些内存中的变量就可以让子进程使用。若父进程用export功能，就可以让自定义的变量的内容写到上述内存区域中(环境变量)，当加载另一个子进程时(离开父进程),子shell可以调用父进程的环境变量所在内存区域到自己的环境变量内存区域中

语系 locale

查看语系

locale

查看支持的所有语系

locale -a #没有的话可以安装语言包

修时修改语系

LANG=en\_US.utf8

永处修改要写到配置文件去

cat /etc/locale.conf

LANG=en\_US.utf8

read 读取来自键盘输入的变量内容

-p 后面接提示字符

-t 接等待的秒数

read -p "please enter your name: " -t 30 name -> hello -> echo $name -> hello

declare 声明变量的类型

-a 将后面的变量定义为数组(array)类型

-i 将后面的变量定义为整数类型

-x 用法与export一样，将后面的变量变成环境变量

-r 交后面的变量设置成为readonly，不可更改内容与不能unset

-P 单独列出变量的类型

declare -i sum=100+200+300 -> echo $sum -> 600 #变量类型默认为字符串所以要声明为整数类型,BASH的数值运算中,默认最多仅能到达整数形态，所以1/3结果是0

数组

设置数组 var[index]=content

var[1]="small min"

var[2]="big min"

var[3]="nice min"

echo ${var[1]} -> "small min"

echo "${var[1]}, ${var[2]}, ${var[3]}"

ulimit

可以限制用户某些资源，如可以开启的文件数量,可以使用的cpu时间，使用的内存总量等

-a 列出所有的限制额度

-f 此shell可以建立的最大文件容量，单位为kbytes

变量的删除与替换

[root@localhost ~]# echo ${path}

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin

删除/usr/local/sbin 第一个目录

[root@localhost ~]# echo ${path#/\*local/sbin:}

/usr/local/bin:/usr/sbin:/usr/bin:/root/bin

删除前面所有的目录只保留 /root/bin

[root@localhost ~]# echo ${path##/\*bin:}

/root/bin

# 删除符合文字的最短那一个

## 删除符合文字的最长那一个

删除最后一个目录

[root@localhost ~]# echo ${path%:\*bin}

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin

只保留第一个目录

[root@localhost ~]# echo ${path%%:\*bin}

/usr/local/sbin

% %% 与# ##类似，但是从后面匹配

${变量/旧字符/新字符} 替换符合的第一个旧字符

￥{变量//旧字符/新字符} 全部替换

20200519

prtg添加https域名监控,add sensor要选择HTTP Advanced类型，然后域名填http://abc.com就行,Require Keyword要选择Set sensor to error if keyword is missing,对应的字符串可以填要监控的字符串(检测有没有劫持),也可以随便填比如(abc.com)

变量的测试与内容转换

root@localhost ~]# echo $username

username变量没有设置时

[root@localhost ~]# username=${username-root}

[root@localhost ~]# echo $username

root

变量为空时

[root@localhost ~]# username=""

[root@localhost ~]# username=${username-root}

[root@localhost ~]# echo $username

[root@localhost ~]# username=${username:-root}

[root@localhost ~]# echo $username

root

其它先跳过

命令别名

alias, 可以将常用或长的命令创建成一个新的命令

alias lm='ls -al|mort' #

也可以替换命令

alias rm='rm -i' #这样rm就变成了新的有-i参数的命令

alias 直接列出所有别名

unalias 取消别名

历史命令

history

-n 列出最近n条命令

-c 将目前的shell中的所有history内容全部清除

-w 将目前的history记录到histfiles中

当以bash登录后,系统会主动同~/.bash\_history读取以前执行

过的命令。~/.bash\_history能记录的数量与HISTFILESIZE变量有关.假设这次登录后执行了100条命令,那当注销时,系统会将101~1100更新到~/.bash\_history中，也就是说注销时会把最近的HISTFILESIZE记录到记录文件中,也可以用history -w强制写入

多用户登录命令写入为最后注销那个

bash shell的操作环境

路径与命令查找顺序

1, 相对/绝对路径执行命令, 如/bin/ls

2, 由alias找到该命令执行

3, 由bahs内置的(builtin)命令来执行

4, 通过$PATH变的的顺序找到的第一个变量来执行

可以通过type -a ls 查看顺序

bash登录与欢迎信息

/etc/issue #登录就看到的欢迎信息如操作系统与硬件等级，可以修改

/etc/motd #可以设置让用户登录就看到的信息,如某时间点系统维护

bash的环境配置文件

一进入Bash就会取得一些有用的变量是因为系统有一些环境配置文件，让bash启动时就直接读取这些文件，这些环境配置文件又分为全局系统配置文件与文人偏好配置文件，alias与自定义的变量在bash注销时会失效，要想保留就得写得配置文件中

login shell 与non-login shell

区别在于有没有登录

login shell

取得bash时需要完整的登录流程,如tty1~tty6登录需要账密,些时取得的bash称为login shell

non-login shell

如在X的图形界面中启动终端或在原本的bash环境下再次输入bash命令,都是不需要账密的

login shell与non-loginshell读取的配置文件不一样

一般来说,login shell只会读取两个配置文件

/etc/profile, 系统的整体设置，最好不要修改

~/.bash\_profile 或~/.bash\_login 或 ~/.profile 属于用户个人设置,用户要添加的自己的数据就写入这里

/etc/profile(login shell才会读)

可以利用用户的标识符(uid)在、来决定很多重要的变量数据,也是每个用户登录取得bash时一定会读取的配置文件，想要帮所有用户设置整体环境就是改这里,主要变量有

PATH

MAIL

USER

HOSTNAME

HISTSIZE

umask

/etc/profile还会云调用外部的配置文件，下面的文件会依序被调用

/etc/profile.d/\*.sh

/etc/locale.conf #语系相关

/usr/share/bash-comletion/completion/\* #命令与文件补齐等

默认情况下读取的整体环境配置文件其实只有/etc/profile，但/etc/profile会调用其它的配置文件

读完整体环境设置后就会读取用户的个人配置文件,一般是这三个

/.bash\_profile

或~/.bash\_login

或 ~/.profile

只实只会读取其中的一个,读取顺序从上到下，前面的不存在才会云读取下面的

[leison@localhost ~]$ cat ~/.bash\_profile

# .bash\_profile

# Get the aliases and functions

if [ -f ~/.bashrc ]; then

. ~/.bashrc

fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH

如PATH变量有设置，以累加的方式并export,-f ~/.bashrc则是存在bashrc就读入bashrc的配置，. ~/.bashrc 等于sorce ~/.bashrc,也就是说~/.bash\_profile其实还会再调用~/.bashrc的内容，所以最终读取的文件是~/.bashrc,可以将自己的偏好设置写入该文件

source 配置文件

配置文件/etc/profile等都是在取得login shell后才会读取的配置文件，所以如果将自己的偏好设置写入上述文件，那就要注销再登录后设置才会生效.source可以直接读取配置文件而不再注销登录

source /etc/profile

. /etc/profile #两条命令等效

non-login shell

该bash只会读取~/.bashrc而已

~/.bashrc

[root@localhost ~]# cat ~/.bashrc

# .bashrc

# User specific aliases and functions

alias rm='rm -i'

alias cp='cp -i'

alias mv='mv -i'

# Source global definitions

if [ -f /etc/bashrc ]; then

. /etc/bashrc

fi

root的bashrc已规范好命令别名的安全选项,centos 7.x还会主动调用/etc/bashrc文件,因为/etc/bashrc帮我们的BASH定义下面内容

根据不同的uid设置umask的值

根据不同的uid设置提示字符(PS1变量)

调用/etc/profile.d/\*.sh设置

其它相关

/etc/man\_db.conf

规范使用man page的该去哪查看数据的路径,tarball安装的软件的命令帮助man可能需要修改

~/.bash\_history

登录bash后读取此文件到内存，在BASH内就可以看到历史命令了

~/.bash\_logout

注销bash后,系统再帮我做什么操作后才离开,如清屏

终端的环境设置 stty set

一般不修改,本身设置已经很好

stty erase ^h #用ctrl + h 删除字符(原来是Backspace)

set

-u,默认不启用，若启用，当使用未设置变量时，会显示错误信息

echo $- # $-变量内容就是set的所有设置

set -u

echo $virbird # 期待virdbird变量不存在,则会报错，没启用-u时会显示空

BASH默认的组合键

ctrl + c 终止目前的命令

ctrl + d 输入结束EOF，例如由件结束的时候

ctrl + m 回车

ctrl + s 停止屏幕输出

ctrl + q 恢复屏幕输出

ctrl + u 在提示符下将整行命令删除

ctrl + z 暂停目前的命令

通配符与特殊符号

通配符

\* 代表0到无穷个任意字符

? 代表一定有一个任意字符

[] 代表一定有一个在括号内的字符(非任意字符),如[[abcd],代表一定有a或b或c或d的中的任意一个字符

[ - ] 代表在编码顺序内的所有字符,如[0-9]代表0和9之间的所有数字

[^ ] ^反向选择, [^abc]代表一定有一个字符,但不是a,b,c

例找出/etc下面以cron为开头的文件

ll -d /etc/cron\*

找出/etc下面刚好是5个字母的文件名

ll -d /etc/?????

找出/etc下面含有数字的文件

ll -d /etc/\*[0-9]\*

找出/etc下面文件名开头为非小写字母开头的文件

ll -d /etc/[^a-z]\*

特殊符号

# 注释符号，写脚本常用

\ 转义符,将特殊字符与通配符还原成一般符号

| 管道

; 连续命令执行分隔符

~ 用户家目录

$ 使用变量前导符

!　逻辑运算上的非(not)

/ 路径分隔符，目录符号

>, >> 重定向输出

<, << 重定向输入

'' 单引号,不具备变量替换功能,单引号内的$变量为纯文本

"" 双引号,具备变量替换功能

`` 反单引号,`中间的命令可先执行，等于$()

( ) 在中间为子shell的起始与结束

{ } 在中间为命令区块的组合

数据流重定向

standard output 与 standar error output

标准输出指的是命令执行所返回的正确信息,标准错误输出指的是命令执行失败返回的错误信息

标准输入(sdin): 代码为0, 使用< 或 <<

标准输出(stdout): 代码为1， 使用> 或>> # 1> 1不写也行

标准错误输出(stderr): 代码为2, 合用 2> 或2>>

ll / > test # 把ll / 输出的信息写入文件或设备, 若test不存在系统则自动建立

1> 以覆盖的方法将正确的数据输出到指定的文件或设备上

1>> 以累加的方法将正确的数据输出到指定的文件或设备上

2> 以覆盖的方法将错误的数据输出到指定的文件或设备上

2>> 以累加的方法将错误的数据输出到指定的文件或设备上

注意 1>> 或 2>> 中间是没有空格的

[ABC@localhost ~]$ find /home -name .bashrc

/home/leison/.bashrc

find: ‘/home/arod’: Permission denied

find /home -name .bashrc > list\_right 2> list\_error

/dev/null 垃圾黑洞,可以吃掉导向这个设备的任何信息

[leison@localhost ~]$ find /home -name .bashrc 2> /dev/null

/home/leison/.bashrc #正确的输出屏幕,错误写出黑洞

将正确与错误的信息都写list

find /home -name .bashrc >list 2>&1 #stderro输出到stout,2的转到1

find /home -name .bashrc &> list 两条都可以

standard input <与<<

$ cat > test

hello

#ctrl + d 退出

cat test

hello

$ cat > test < ~/.bashrc #用文件替代键盘的输入

cat test

# .bashrc

# Source global definitions

if [ -f /etc/bashrc ]; then

. /etc/bashrc

.....

.....

$ cat > test << "eof"

> hello

> good

> eof

[leison@localhost ~]$ cat test

hello

good

利用<<可以不用ctrl + d，对写脚本有帮助

命令的执行的判断根据 ; && ||

cmd;cmd (不考虑命令相关性的连续命令执行)

如 $ sync;sync;shutdown -h now #关机前先执行两次sync同步写入磁盘操作

$? (命令返回值)与 && 或 ||

若前一个命令执行的结果为正确,在Linux下面会返回一个$?=0的值

cmd1 && cmd2, 若cmd1执行完毕且正确执行($?=0),则开始执行cmd2

若cmd1执行完毕且为错误($?不等于0),则cmd2不执行

cmd1 || cmd2, 若cmd1执行完毕且正确执行($?=0),则cmd2不执行

若cmd1执行完毕且为错误($?不等于0),则开始执行cmd2

例若存在/tmp/abc,则建立/tmp/abc/hehe文件

假设存在

ls /tmp/abc && /tmp/abc/hehe

例测试/tmp/abc是否存在,若在则不创建 ，不存在则创建

ls /tmp/abc || mkdir /tmp/abc

例不清楚/tmp/abc是否存在,但就是要建立/tmp/abc/hehe

ls /tmp/abc || mkdir /tmp/abc && touch /tmp/abc/hehe

若存在/tmp/abc则不执行mkdir,$?=0向后转, &&遇到0,执行touch

若不存在/tmp/abc,则执行mkdir,后$?=0向后转,touch照样会执行

例判断/tmp/virding是否存在,存在显示"exist",不存在显示"not exist"

ls /tmp/virding && echo "exist" || echo "not exsit" #顺序重要

20200523

管道命令(pipe)

管道命令仅会处理标准输出,对于标准错误会予以忽略

管道命令必须能够接受来自前一个命令的数据成为标准输入继续输入处理才行

管道命令

less more head tail等

选取命令 cut grep

[root@localhost ~]# echo $PATH

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin

[root@localhost ~]# echo $PATH | cut -d ':' -f 5

/root/bin

[root@localhost ~]# echo $PATH | cut -d ':' -f 3,5

/usr/sbin:/root/bin

[root@localhost ~]# export

declare -x HISTCONTROL="ignoredups"

declare -x HISTSIZE="1000"

declare -x HOME="/root"

declare -x HOSTNAME="localhost.localdomain"

declare -x LANG="en\_US.UTF-8"

.......

[root@localhost ~]# export | cut -c 12-

HISTCONTROL="ignoredups"

HISTSIZE="1000"

HOME="/root"

HOSTNAME="localhost.localdomain"

grep

[root@localhost ~]# last

root pts/0 192.168.1.8 Fri May 22 22:15 still logged in

root pts/1 192.168.1.3 Fri May 22 18:59 - 23:43 (04:43)

root tty1 Fri May 22 18:59 still logged in

root pts/0 192.168.1.5 Fri May 22 11:04 - 20:51 (09:47)

reboot system boot 3.10.0-957.el7.x Fri May 22 10:59 - 03:28 (16:28)

有出现root的一行就找出来

root@localhost ~]# last | grep 'root'

root pts/0 192.168.1.8 Fri May 22 22:15 still logged in

root pts/1 192.168.1.3 Fri May 22 18:59 - 23:43 (04:43)

root tty1 Fri May 22 18:59 still logged in

root pts/0 192.168.1.5 Fri May 22 11:04 - 20:51 (09:47)

取反

[root@localhost ~]# last | grep -v 'root'

reboot system boot 3.10.0-957.el7.x Fri May 22 10:59 - 03:29 (16:30)

reboot system boot 3.10.0-957.el7.x Tue Apr 28 12:37 - 03:29 (24+14:52)

reboot system boot 3.10.0-957.el7.x Sat Apr 18 14:33 - 03:29 (34+12:56)

[root@localhost ~]# last | grep 'root' | cut -d ' ' -f 1

root

root

root

排序命令

sort

-n 以纯数字来排序

-r 反向排序

-t 分隔符,默认[Tab]

-k 指定那个区间来进行排序

普通排序,sort默认以第一条信息来排，且以文字形式排序

[root@localhost ~]# cat /etc/passwd | sort

adm:x:3:4:adm:/var/adm:/sbin/nologin

alex:x:1001:1001::/home/alex:/bin/bash

apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin

arod:x:1002:1002::/home/arod:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

chrony:x:998:996::/var/lib/chrony:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

以第3段且以纯数字来排序

[root@localhost ~]# cat /etc/passwd | sort -t ':' -k 3 -n

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

sync:x:5:0:sync:/sbin:/bin/sync

uniq (去重)

-c 进行计数

使用last取出账号栏,仅取出账号，进行排序后取出一位

[root@localhost ~]# last | cut -d ' ' -f 1| sort|uniq

reboot

root

wtmp

加多账号登录次数

root@localhost ~]# last | cut -d ' ' -f 1 | sort | uniq -c #先sort

1

16 reboot

89 root

1 wtmp #空白与wtmp是默认字符可以忽略

wc (统计行或字符)

-l 统计行数

-m 统计字符

[root@localhost ~]# cat /etc/man\_db.conf |wc

131 723 5171

# 行 字数 字符数

统计登录次数

[root@localhost ~]# last | grep [a-zA-Z]|grep -v 'wtmp'|grep -v reboot | wc -l

89

双向重定向 tee

可以让standoutput转存一份到文件内并将同样的数据在屏幕去处理

-a 以累加的方式将数据加入文件中

[root@localhost ~]# last | tee last.list | cut -d ' ' -f 1

root

root

root

# ls -l | tee ~/homefile |more

ls -l / | tee -a ~/homefile | more

字符转换命令

tr

可以删除字符或替换字符

-d 删除字符

-s 替换字符

所有小写变成大字

last | tr '[a-z]' '[A-Z]'

先把unix文件变成dos换行符文件

cp /etc/passwd ~/passwd && unix2dos ~/passwd

查看换行符

cat -A ~/passwd

root:x:0:0:root:/root:/bin/bash^M$

bin:x:1:1:bin:/bin:/sbin/nologin^M$

daemon:x:2:2:daemon:/sbin:/sbin/nologin^M$

删除^M

cat ~/passwd | tr -d '\r' > ~/passwd.linux

col

-x 将tab键转换成对等的空格键

[root@localhost ~]# cat -A /etc/man\_db.conf

.....

SECTION^I^I1 1p 8...

......

^I为1个tab键

[root@localhost ~]# cat /etc/man\_db.conf |col -x | cat -A

.....

SECTION 1 1p 8 2 3 3p 4 5

......

join

合并两个文件,有相同数据的那一行才将它加在一起

-t join默认以空格分隔数据,并且比对第一栏位

-1 数字1，代表第一个文件要用哪个栏位比对

-2 代表第二个文件要用哪个栏位比对

[root@localhost ~]# head -n 3 /etc/passwd /etc/shadow

==> /etc/passwd <==

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

==> /etc/shadow <==

root:$6$esu2az6WJraoEBnx$4qVMw/P4z4gn2S/RFhDD/SiyY2MZsM6xlfro8EFxrcvldSweyqFcF.OWrm2/vUG/.VXc5a4R2yDioqr1NS3Aw/::0:99999:7:::

bin:\*:17834:0:99999:7:::

daemon:\*:17834:0:99999:7:::

默认以第一栏比对并合并文件,第二文件的第一栏就不显不

[root@localhost ~]# join -t ':' /etc/passwd /etc/shadow | head -n 3

root:x:0:0:root:/root:/bin/bash:$6$esu2az6WJraoEBnx$4qVMw/P4z4gn2S/RFhDD/SiyY2MZsM6xlfro8EFxrcvldSweyqFcF.OWrm2/vUG/.VXc5a4R2yDioqr1NS3Aw/::0:99999:7:::

bin:x:1:1:bin:/bin:/sbin/nologin:\*:17834:0:99999:7:::

daemon:x:2:2:daemon:/sbin:/sbin/nologin:\*:17834:0:99999:7:::

例2

[root@localhost ~]# head -n 3 /etc/passwd /etc/group

==> /etc/passwd <==

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

==> /etc/group <==

root:x:0:

bin:x:1:

daemon:x:2:

[root@localhost ~]# join -t ':' -1 4 /etc/passwd -2 3 /etc/group | head -n 3

join: /etc/passwd:6: is not sorted: sync:x:5:0:sync:/sbin:/bin/sync

join: /etc/group:11: is not sorted: wheel:x:10:

0:root:x:0:root:/root:/bin/bash:root:x:

1:bin:x:1:bin:/bin:/sbin/nologin:bin:x:

2:daemon:x:2:daemon:/sbin:/sbin/nologin:daemon:x:

#第一个文件4第4栏与第二文件的第三栏都是gid所以比对这一栏,相同的栏位被称到第一栏,剩下的第二文件的同行补到第一文件，注意join前所需处理的文件应该要事先排序

paste

也是将两个文件合并，直接将两行巾在一起，且以tab键隔开，不需要比对两个文件的数据

-d 可接分隔符,默认tab的数据

- file部分写成-，表示来自标准输入

root@localhost ~]# paste /etc/passwd /etc/group

root:x:0:0:root:/root:/bin/bash root:x:0:

bin:x:1:1:bin:/bin:/sbin/nologin bin:x:1:

daemon:x:2:2:daemon:/sbin:/sbin/nologin daemon:x:2:

group的内容作为标准输入给paste

[root@localhost ~]# cat /etc/group | paste /etc/passwd /etc/shadow - | head -n 3

root:x:0:0:root:/root:/bin/bash root:$6$esu2az6WJraoEBnx$4qVMw/P4z4gn2S/RFhDD/SiyY2MZsM6xlfro8EFxrcvldSweyqFcF.OWrm2/vUG/.VXc5a4R2yDioqr1NS3Aw/::0:99999:7::: root:x:0:

bin:x:1:1:bin:/bin:/sbin/nologin bin:\*:17834:0:99999:7::: bin:x:1:

expand

将tab按键转成空格键

-t 后面接数字,一般是一个tab可以用8个空格键替换，可以自定义

[root@localhost ~]# grep '^MANPATH' /etc/man\_db.conf |head -n 3 | expand -t 6| cat -A

MANPATH\_MAP /bin /usr/share/man$

MANPATH\_MAP /usr/bin /usr/share/man$

MANPATH\_MAP /sbin /usr/share/man$

已将tab转成6个空格

unexpand 可以将空格转为tab键

split

划分命令

-b 后接谷划分的文件大小，单侠,b,k,m等

-l 以行数来划分

[root@localhost tmp]# split -b 300k /etc/services services

[root@localhost tmp]# ll

total 656

drwxr-xr-x. 2 root root 18 May 22 22:36 abc

-rw-r--r--. 1 root root 307200 May 23 22:47 servicesaa

-rw-r--r--. 1 root root 307200 May 23 22:47 servicesab

-rw-r--r--. 1 root root 55893 May 23 22:47 servicesac

services，划分的文件名任取，被划分的取名xxxaa,xxxab,xxxac等

将上面三个划分的文件合成一个文件

root@localhost tmp]# cat servicesa\* >> serviceback

[root@localhost tmp]# ll

total 1312

drwxr-xr-x. 2 root root 18 May 22 22:36 abc

-rw-r--r--. 1 root root 670293 May 23 22:48 serviceback

-rw-r--r--. 1 root root 307200 May 23 22:47 servicesaa

-rw-r--r--. 1 root root 307200 May 23 22:47 servicesab

-rw-r--r--. 1 root root 55893 May 23 22:47 servicesac

例ls -al / 的信息每10行划分一个文件

[root@localhost tmp]# ls -al / | split -l 10 - lsroot

[root@localhost tmp]# wc -l lsroot\*

10 lsrootaa

10 lsrootab

8 lsrootac

28 total

重点是-, 如果需要输入或输出时没有文件,-就会被当成stdin或stdout

xargs(参数代换)

很多命令其实不支持管道命令,可以通过xargs来提供命令使用,标准输入

-0 如果输入的stdin含有特殊字符,如 ` \ 空格等时,-0可以将它还原成一般字符

-e EOF的意思,后面可以接一个字符,当xargs分析到这个字符时,就会停止工作

-p 在执行每个命令时,都会询问使用者

-n 后面接次数,每次command命令执行时,要使用几个参数的意思

cut -d ':' -f 1 /etc/passwd | head -n 3 | xargs -n 1 id

uid=0(root) gid=0(root) groups=0(root)

uid=1(bin) gid=1(bin) groups=1(bin)

uid=2(daemon) gid=2(daemon) groups=2(daemon)

#因为id后面只能接一个参数,所以要-n 1 来一个一个参数给id，不然直接给3个出出错

加多询问

cut -d ':' -f 1 /etc/passwd | head -n 3 | xargs -p -n 1 id

id root ?...y

uid=0(root) gid=0(root) groups=0(root)

id bin ?...y

uid=1(bin) gid=1(bin) groups=1(bin)

id daemon ?...y

将所有的/etc/passwd内的账号都以id来查看,但查到sync时就结束命令

$ cut -d ':' -f 1 /etc/passwd | xargs -e'sync' -n 1 id

uid=0(root) gid=0(root) groups=0(root)

uid=1(bin) gid=1(bin) groups=1(bin)

uid=2(daemon) gid=2(daemon) groups=2(daemon)

uid=3(adm) gid=4(adm) groups=4(adm)

uid=4(lp) gid=7(lp) groups=7(lp)

#注意-e后面没空格,因为sync是在第6行，-e为eof，查到sync就结束命令(包括sync那行)

找出/usr/sbin下的有特殊权限的文件并使用ls -l列出

find /usr/sbin -perm /7000 | xargs ls -l

-rwsr-xr-x. 1 root root 117504 Nov 8 2018 /usr/sbin/mount.nfs

-rwxr-sr-x. 1 root root 7208 Oct 31 2018 /usr/sbin/netreport

-rwsr-xr-x. 1 root root 11216 Apr 11 2018 /usr/sbin/pam\_timestamp\_check

-rwxr-sr-x. 1 root postdrop 218632 Oct 31 2018 /usr/sbin/postdrop

-rwxr-sr-x. 1 root postdrop 260112 Oct 31 2018 /usr/sbin/postqueue

-rwsr-xr-x. 1 root root 36280 Apr 11 2018 /usr/sbin/unix\_chkpwd

-rwsr-xr-x. 1 root root 11376 Oct 31 2018 /usr/sbin/usernetctl

#ls 并不是管道命令所以要用xargs替代，也可用下面的命令

ls -l $(find /usr/sbin -perm /7000)

- （减号的用途)

mkdir /tmp/homeback

tar -cvf - /home | tar -xvf - -C /tmp/homeback

#tar打包并不是记录下来而是转送到stdout,后面的-则是使用前一个命令的stdout,这样就不需要文件名了

正则表达式(regular expression)

处理字符串的方法,它以行为单位来进行字符串的处理操作，通过一些特殊符号的辅助,可以让用户轻易地完成查找删除替换某特定字符串的处理过程

正则表达式是一种表示法,只要程序支持这种表示法,那该程序就可以用来作为正则表达式的字符串之用，如vi, grep, awk, sed等,像ls ,cp这类不支持正则表达式的程序就只能使用bash自己本身的通配符而已

通配符只是bash操作接口的一个功能,正则表达式是字符串处理的表示方式,两者完全不同

基础正则表达式与扩展正则表达式

依照不同的严谨度而分,扩展正则表达式可以使用()与| 来组合

基础正则表达式

语系对正则表达式的影响

文字编码系统里文件其实记录的只有0和1,我们看到的字符与数字都是通过编码表转换来的。所以不同的语系的编码数据并不相同,所以造成的选取结果可能会有差异.如在英文大小写的编码顺序中, zh\_CN.big5与C这两种语系的输出结果:

LANG=C时,0,1,2,3,4.....A,B,C....a,b,c.....z

LANG=zh\_CN时,0,1,2,3.....,a,A,b,B,c,C.......

如果想选取[A-Z]时,那LANG=C时就可以仅识别到大写字符,而zh\_CN时就会连abc等小写字符一起选 出来，所以语意要留意

特殊符号

[:alnum:] 代表英文大小写字符及数字,0-9,a-z,A-Z

[:alpha:] 代表任何英文大小写字符,A-Z,a-z

[:upper:] 代表大写字符,A-Z

[:lower:] 代表小写字符 a-z

[:digit:] 代表数字 0-9

grep高级用法

-A 后面接数字,after的意思,除了列出该行外,后续的n行也一起列出

-B 后面接数字,before的意思,除了列出该行外,前面的n行也一起列出

-n 显示行号

[root@localhost tmp]# cat lsrootaa | grep 'etc'

drwxr-xr-x. 84 root root 8192 May 22 23:18 etc

[root@localhost tmp]# cat lsrootaa | grep -A 2 'etc'

drwxr-xr-x. 84 root root 8192 May 22 23:18 etc

drwxr-xr-x. 4 root root 32 Apr 29 05:09 home

lrwxrwxrwx. 1 root root 7 Aug 11 2019 lib -> usr/lib

[root@localhost tmp]# cat lsrootaa | grep -B1 'home'

drwxr-xr-x. 84 root root 8192 May 22 23:18 etc

drwxr-xr-x. 4 root root 32 Apr 29 05:09 home

--color=auto #默认已经加入在alias当中了,不需额外添加

基础正则表达式练习

查找包含test或taste的行

grep -n 't[ae]st' regular\_expree.txt # -n显示行号

查找oo前面无g的行

grep -n '[^g]oo' regular\_expree.txt

oo前面不想要有小写字符

grep -n '[^a-z]oo' regular\_expree.txt

#由于小写字符在编码上是顺序的,所以可以用a-z连续表示

取得有数字那一行

grep -n '[0-9]' regular\_expree.txt

由于考虑到语系对于编码的顺序，上面那个例子可以表示为

grep -n '[^[:lower:]]oo' regular\_express.txt

grep -n '[[:digit:]]' regular\_express.txt

行首^与行尾$

在行首列出the

grep '^the' regular\_express.txt

开头是小写字符

grep -n '^[a-z]' regular\_expree.txt

开头不是英文字母

grep -n '^[^a-zA-Z]' regular\_express.txt

行尾.结束

grep -n '\.$' regular\_express.txt #小数点.有其它意义,所以要转义

找出空白行

grep -n '^$' regular\_express.txt

忽略空白行与#开头的行

cat /etc/rsyslog.conf | grep -v '^#'| grep -v '^$'

任意一个字符 . 与重复字符\*

. 代表一定有一个任意字符

\* 代表重复前面一个字符0到无穷多次

#在bash中\*可以代表0到多个字符,两者不同

找出g??d字符

grep 'g..d' r.txt

至少两个o以字

grep 'ooo\*' r.txt

g开头与g结尾的字符串,中间可有可无

grep 'g.\*g' r.txt # .\* 代表0或多个任意字符

找出任意数字的行

grep '[0-9][0-9]\*' r.txt

# grep '[0-9]' r.txt 也可以

限定连续字符区间 {}

因为{}在shell中有特殊意义,所以必须要使用转义符 \ 来让它失去意义

找到两个o的字符

grep 'o\{2\}' r.txt

g后面接2到5个o，再g

grep 'go\{2,5\}g' r.txt

两个o以上

grep 'go\{2,\}g' r.txt

# grep 'gooo\*' r.txt 也可以

基础正则表达式字符集合

通过以上的例子可以知道集合如下

^word 待查找的字符在行首

word$ 在行尾

. 代表一定有一个任意字符

\ 转义符,将特殊符号的特殊意义去除

例 要找含有单引号 ' 的一行

grep \' regular\_express.txt #注意不能用双单引号结合

[list] 列出想要的字符,例

grep 'g[ld]' r.txt 则字符串可以是 gl, gd

[n1-n2] 列出想要的字符

grep '[A-Z]' r.txt

[^list] 取反

grep 'oo[^t]' r.txt 可以找到ooa,oog等,但不能oot

\{n\} 连续n个前一个re字符

\{n,m\} 连续n到m个前一个re字符

\{n,\} 连续n个以上前一个re字符

sed

管道命令,可以将数据进行替换,删除,新增,选取,选特定行等功能

-n 使用安静模式,一般sed中,所有来自stdin的数据一般都会被列到屏幕上,加上-n后,则只有经过sed特珠处理的那一行(或操作)才会被列出来

-e 直接在命令行模式上进行sed操作

-f 直接将sed的操作写在一个文件内

-r sed的模式使用扩展正则表达式,默认是基础正则表达式

-i 直接修改读取的文件内容,不在屏幕输出

操作说明 n1,n2 function

n1,n2 不见得会存在,一般代表选择进行的行数,如操作10~20行， 10,20 操作行为

function有如下行为

a 新增,后面可以接字符,而这些字符会在新的一行出行(下一行)

c 替换,后面可以接字符,这些字符可以替换n1,n2之间的行

d 删除,因为是删除所以d后面不接任何东西

i 插入,后接字符,在新的一行出现(上一行)

p 打印,将某个选择的数据打印出来,通常p会与参数 sed -n 一起运行

s 替换,通常搭配正则表达式,如 1,20s/old/new/g

以行为单行的新增/删除功能

将/etc/passwd的内容列出并打印行号，且将2~5行删除

nl /etc/passwd | sed '2,5d'

1 root:x:0:0:root:/root:/bin/bash

6 sync:x:5:0:sync:/sbin:/bin/sync

7 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

#sed后面接的操作要用''包住

删除第3行到最后一行

nl /etc/passwd | sed '3,$d'

在第2行后加上字符

nl /etc/passwd | sed '2a drink tea'

在第2行前加字符

nl /etc/passwd | sed '2i drink tea'

在第2行后面加入两行

nl /etc/passwd | sed '2a drink tea........\

> or drin beer'

#用 \

以行为单位的替换与显示功能

替换第2~5行的内容

nl /etc/passwd | sed '2,5c no\_nombers'

只显示5~7行

nl /etc/passwd | sed -n '5,7p'

部分数据的查找并替换功能

提取ip

ifconfig | grep 'inet'

inet 192.168.1.12 netmask 255.255.255.0 broadcast 192.168.1.255

inet6 fe80::b767:af5c:a4e8:43b5 prefixlen 64 scopeid 0x20<link>

....

ifconfig | grep '192.168.1.12'|sed 's/.\*inet \*//g'|sed 's/ \*net.\*$//g'

192.168.1.12

/etc/man\_db.conf文件下找出MAN行,不要#开头注释与空白行

cat /etc/man\_db.conf | grep 'MAN' | sed 's/^#.\*//g' | sed '/^$/d'

#注意#.\*,要连同#后面的数据也删除掉,如只有#则只删除#

直接修改文件内容

将regular\_express.txt以每一行结尾为.的行改为!

sed -i 's/\.$/\!/g' regular\_express.txt

最后一行新增字符#this is test

sed -i '$a #this is test' regular\_express.txt

扩展正则表达式 egrep

+ 代表重复一个或一个以上的前一个字符

egrep 'go+d' r.txt

? 代表0个或1个前一个字符

egrep 'go?d' r.txt

| 用或or的方式找出数个字符串

egrep 'go|good' r.txt

() 找出群组字符串

egrep 'g(la|oo)d' r.txt

可以找出glad或good

()+ 多个重复群组

echo 'AxyzxyzxyzxyzC' | egrep 'A(xyz)+C' #注意没空格

文件的格式化

printf '%10s\t %5i\t %5i\t %5i\t %8.2f\t \n' $(cat p.txt|grep -v 'Name' )

DmTsai 80 60 92 77.33

VBird 75 55 80 70.00

Ken 60 90 70 73.33

#对这三行格式化,%10s代表10个字符长度的字符串,\t tab， %5i, 5个字符长度的整数，%8.2f 8个字符长度的浮点数2位小数,小数点占一个字符长度 \n 输出新的一行

[root@localhost ~]# printf '\x45\n'

E

#printf可以输出ascall数字与字符对应，上面就是输出16进字的45是什么字符

awk 数据化处理工具

sed常常用于整行的处理,awk倾向于一行当中分数个字段来处理

模式

awk '条件类型1{操作1} 条件类型2{操作2} ...' filename

awk后续可以接文件,也可以读取来自前一个命令的标准输出。主要是处理每一行的字段内的数据,默认字段分隔符为'空格键或tab'

[root@localhost ~]# last -n 5 | awk '{print $1 "\t" $3}'

root desktop-kacbmdc

root desktop-kacbmdc

root 192.168.1.5

root 192.168.1.8

root 192.168.1.8

wtmp Sun

以上为awk最常用的操作,通过print功能将字段列出来,这里我们指定分隔符\t(tab), ',' 为空格分隔,因为不论每一行都要处理，所以就不需要'条件类型',然后我们要的是第1与第3栏

awk每一行每个字段都有变量名称,第1栏, $1, 第2栏, $2.... $0代表一整行

整个awk的处理流程是

1 读入第1行,并将第1行的数据写入$0,$1,$2.....变量中

2 根据条件类型,判断是否需要进行后面的'操作'

3 完成所有操作与条件类型

4 若后面还有'行操作',则重复上在1~3步骤,直到所有的数据都读完为止

这样可以知道awk以一行为处理的单位,以字段为最小的处理单位

变量 代表意义

NF 每一行$0拥有的字段数

NR 目前awk所处理的第几行数据

FS 目前空隔符,默认空格键

注意awk后续的所有有操作以单引号'括住,由于单引号与又引号必须成对出现,所以非变量部分要用双引号定义出来,因为单引号已经是awk的命令固定用法了

last后列出每一行的账号,列出目前处理的行数,并且说明该行有多少字段

last -n 5 | awk '{print $1 "\t line:" NR "\tcout numbers: "NF}'

root line:1 cout numbers: 10

root line:2 cout numbers: 10

root line:3 cout numbers: 10

root line:4 cout numbers: 10

root line:5 cout numbers: 10

line:6 cout numbers: 0

wtmp line:7 cout numbers: 7

#注意NF等变量要用大写且不用$符号

awk的逻辑运算符

> 大于

< 小于

>= 大于或等于

<= 小于或等于

== 等于

!= 不等于

注意==是用来判断,如果是要给予一个值用=

例/etc/passwd列出第1列账号与第3例uid且uid小于10

awk '{FS=":"}$3<10{print $1 "\t" $3}' /etc/passwd

root:x:0:0:root:/root:/bin/bash

bin 1

daemon 2

adm 3

....

#结果发现第一行没有正确显示,因为读入第一行行变量$1$2等默认的分融符还是空格,虽然有定义FS=':'，但是第二行才生效,所以要预先用BEGIN设置

[root@localhost ~]# cat /etc/passwd | awk 'BEGIN {FS=":"} $3<10 {print $1 "\t" $3}'

root 0

bin 1

daemon 2

adm 3

lp 4

sync 5

shutdown 6

halt 7

mail 8

[root@localhost ~]# cat t.txt

name lst 2nd 3th

vbird 23000 24000 25000

hello 26000 27000 29000

nice 22000 28000 20000

[root@localhost ~]# cat t.txt | awk 'NR==1{printf "%10s\t %10s\t %10s\t %10s\t %10s\n",

> $1,$2,$3,$4,"Total"}

> NR>=2{total=$1+$2+$3+$4

> printf "%10d\t %10d\t %10d\t %10d\t %10.2f\n",$1,$2,$3,$4,total}'

name lst 2nd 3th Total

0 23000 24000 25000 72000.00

0 26000 27000 29000 82000.00

0 22000 28000 20000 70000.00

#所有awk操作都在{}内,如多个命令辅助可以用;或直接以Enter按键隔开命令并非利用\,格式化输出时printf要用\n来分行,变量可以在awk中直接用,不需$

awk的操作内{}也支持if条件,如上面的NR==1可以写{}内 {if(NR==1)},建议NR==1在外面统一性

文件比对工具

什么时候会用到文件比对,通常是同一软件包的不同版本之间,比较配置文件与原始文件的差异，通常是用在ASCALL纯文本文件的比对

diff

比较两个文件的差异,以行为单位来比对,一般用在ascall纯文本文件比对,通常用在同一文件或软件的新旧版本差异上

cat /etc/passwd | sed -e '4d' -e '6c no sixlines' > /tmp/testpw/newpasswd

#sed 后面接两个以上操作时，每个命令前要-e

diff /etc/passwd /tmp/testpw/newpasswd

4d3

< adm:x:3:4:adm:/var/adm:/sbin/nologin #左边的第4行被删除了,基准右边的是第3行

6c5

< sync:x:5:0:sync:/sbin:/bin/sync

---

> no sixlines #左边的第6行被替换成右边的第5行

不要用diff比对两个不相干的文件

diff还可以比对不同目录下相同文件名的内容

cmp

比对非纯文本,以字节去比对

patch

diff -Naur oldpasswd newpasswd > passwd.patch

cat passwd.patch

--- oldpasswd 2020-05-26 04:51:41.306333925 +0800

+++ newpasswd 2020-05-26 04:22:00.064392202 +0800

@@ -1,9 +1,8 @@ #新旧文件修改界定范围,旧1-9行,新1-8行

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

-adm:x:3:4:adm:/var/adm:/sbin/nologin #左侧文件删除行

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

-sync:x:5:0:sync:/sbin:/bin/sync #左侧文件删除行

+no sixlines #右侧文件新增行

shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

halt:x:7:0:halt:/sbin:/sbin/halt

mail:x:8:12:mail:/var/spool/mail:/sbin/nologin

#上面意思old文件要变成新文件就是第4行要删除与第6行要替换

新制出出来的patch文件用来更新旧文件

patch -p0 <passwd.patch

恢复旧文件内容

patch -R -p0 < passwd.patch

#-p是后面接几层的意思,因为是在同一目录下面比对,因此不需要减去目录

pr

输出文件时顺便打印标题或面码

如 pr /etc/man\_db.conf

重要习题

找出/etc下含有\*的文件或内容

grep '\\*' $(find /etc -type f)

或

find /etc -type f | xargs grep '\\*'

找出/ 下含有\*的文件或内容

grep '\\*' $(find / -type f)

-bash: /usr/bin/grep: Argument list too long #命令串长度有限制,得用xargs

find / -type f 2> /dev/null| xargs -n 10 grep '\\*'

20200530

shell脚本

注意事项

命令从上而下,从左而右地分析与执行

命令,选项,与参数间的多个空格都会被忽略掉

空白行也被忽略掉,tab所产生的空白同样视为空格

如果读取到一个Enter符号(CR),就尝试开始执行该行(命令)

如果一行的内容太多,则可以使用\Enter来扩展至下一行

#注释,任何接在#后面的内容都视作为注释文字而忽略

假设有/home/good/shell.sh命令,执行方法有

直接命令执行,shell.sh必须有rx权限,然后

绝对路径执行 /home/good/shell.sh

或相对路径执行 ./shell.sh

将shell.sh放入PATH指定目录内,如~/bin, 直接执行shell.sh

以bash程序来执行,通过bash shell.sh或 sh shell.sh执行(只需r权限)

#sh可以执行是因为/bin/sh其实是/bin/bash链接文件,用bash是代表要直接以bash的功能执行shell.sh文件内的相关命令,sh有-n或-x参数来检查与跟综shell.sh语法的正确性

脚本范例

#!/bin/bash

#program

# this program show "hello world" in your screen

#History:

# 20200530 leison first release

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin

export PATH

echo -e "Hello world \a \n "

exit 0

1 因为我们使用的是bash,所以必须要用#!/bin/bash来声明文件内使用bash语法,以#!开头的行称为shebang行,那当这个程序执行时,就能够加载bash的相关环境配置文件(一般是非登录shell的~/.bashrc),并执行bash让我们下面的命令能够执行

2 程序内容说明,#用来注释说明,可以一些基本数据,脚本用途日期作者等

3 主要环境变量的声明,PATH或LANG等,这样程序就可以直接执行一些外部命令,而不必写绝对路径

4 主要程序

5 执行结果告知,定义返回值

执行sh shell.sh或chmod +x shell.sh;./shell.sh

简单脚本示例

交换示脚本

#!/bin/bash

read -p "please enter your first name: " firstname

read -p 'please enter your last name: ' lastname

echo -e "\nyou full name is $firstname $lastname"

创建三个以今天昨天前天日期名的空文件

#!/bin/bash

# enter a file filename

read -p 'please enter a filename: ' fileuser

#check filename exit or not,避免用户按到空格键或回车键

filename=${fileuser:-"filename"} #fileuser变量如果存在就把其变量内容给filename,否则把filename内容给filename变量

date1=$(date --date='1 days ago' +%F)

date2=$(date --date='2 days ago' +%F)

date=$(date +%F)

file1=${filename}${date1}

file2=${filename}${date2}

file=${filename}${date}

touch $file1

touch $file2

touch $file

数值运算

#!/bin/bash

read -p "pleas enter a number: " num1

read -p "pleas enter second number: " num2

total=$(( ${num1} \* ${num2} ))

echo "num1 \* numb = $total"

#var=$((运算内容)),也可以declare -i total=${num1}\*${num2},建议$(()),方便记忆而且小括号内可以加上空格符,而且bash shell默认仅支持到整数而已

取余echo $((13%3))

1

echo "12.3\*5" | bc #含有小数点可以通过bc协助,但只对+-\*有效,而且不能用$(())

61.5

通过bc求PI(圆周率),略过,可以设置多位小数值测试cpu负载

脚本执行方式的差异

利用直接执行的方式(绝对路径，相对路径, bash sh等)都是新的bash环境中(子进程)运行的,子进程完成后,子进程内的各项变量或操作将会结束而不会传回父进程中

source或. 执行脚本,将会直接在父进程执行,脚本内的变量或操作在父进程中也是存在的,和source ~/.bashrc道理一样,不能 bash ./barshrc

判断式

test

测试的标志 代表意义

1. 关于某个文件名的『文件名型』判断，如test -e filename 表示存在否

-e 该『文件名』是否存在？(常用)

-f 该『文件名』是否存在且为文件(file)？(常用)

-d 该『文件名』是否存在且为目录(directory)？(常用)

-b 该『文件名』是否存在且为一个block device 装置？

-c 该『文件名』是否存在且为一个character device 装置？

-S 该『文件名』是否存在且为一个Socket 文件？

-p 该『文件名』是否存在且为一个FIFO (pipe) 文件？

-L 该『文件名』是否存在且为一个连结档？

2. 关于文件的权限侦测，如test -r filename 表示可读否(但root 权限常有例外)

-r 侦测该文件名是否存在且具有『可读』的权限？

-w 侦测该文件名是否存在且具有『可写』的权限？

-x 侦测该文件名是否存在且具有『可执行』的权限？

-u 侦测该文件名是否存在且具有『SUID』的属性？

-g 侦测该文件名是否存在且具有『SGID』的属性？

-k 侦测该文件名是否存在且具有『Sticky bit』的属性？

-s 侦测该文件名是否存在且为『非空白文件』？

3. 两个文件之间的比较，如： test file1 -nt file2

-nt (newer than)判断file1 是否比file2 新

-ot (older than)判断file1 是否比file2 旧

-ef 判断file1 与file2 是否为同一文件，可用在判断hard link 的判定上。主要意义在判定，两个文件是否均指向同一个inode

4. 关于两个整数之间的判定，例如test n1 -eq n2

-eq 两数值相等(equal)

-ne 两数值不等(not equal)

-gt n1 大于n2 (greater than)

-lt n1 小于n2 (less than)

-ge n1 大于等于n2 (greater than or equal)

-le n1 小于等于n2 (less than or equal)

5. 判定字串的资料

test -z string 判定字串是否为0 ？若string 为空字串，则为true

test -n string 判定字串是否非为0 ？若string为空字串，则为false。

注： -n亦可省略

test str1 == str2 判定str1 是否等于str2 ，若相等，则回传true

test str1 != str2 判定str1 是否不等于str2 ，若相等，则回传false

6. 多重条件判定，例如： test -r filename -a -x filename

-a (and)两状况同时成立！例如test -r file -a -x file，则file 同时具有r 与 x 权限时，才回传true。

-o (or)两状况任何一个成立！例如test -r file -o -x file，则file 具有r 或 x 权限时，就可回传true。

! 反相状态，如test ! -x file ，当file 不具有x 时，回传true

OK！现在我们就利用test 来帮我们写几个简单的例子。首先，判断一下，让使用者输入一个文件名，我们判断：

这个文件是否存在，若不存在则给予一个『Filename does not exist』的讯息，并中断程式；

若这个文件存在，则判断他是个文件或目录，结果输出『Filename is regular file』或 『Filename is directory』

判断一下，执行者的身份对这个文件或目录所拥有的权限，并输出权限资料！

#!/bin/bash

read -p "please enter a filename: " file

#判断输入是否为空,为空就退出

test -z $file && echo "you need enter a name" && exit

#判断是否存在,不存在退出

test ! -e $file && echo "not exit " && exit

#判断是文件还是目录

test -f $file && echo "$file is a file" || echo "$file is a direct"

test -r $file && echo "$file have r "

test -w $file && echo "$file have w "

test -x $file && echo "$file have x "

[] 和test一样

[ -e ${HOME} ]; echo $?

0

[ "${HOME}" == "${MAIN}" ]

#注意空格,其实=与==都可以，但bash中=是变量设置,==逻辑判断,建议最好==

中括号[]内的每个组件都需要有空格分隔

在中括号内的变量,最好都以双引号括起来

在中括号内的常数,最好都以单或双引号括起来

test用法与[]几乎一模一样,中括号比较常用在条件判断式If...then...fi中

例执行一个程序让用户选择Y或y,当选择y或Y时,显示OK,continue.当选择n或N时,显示Oh,interrupt,选择其它显示I don't know what your choice is

#!/bin/bash

read -p "please enter y/Y or n/N : " num

[ "$num" == "y" -o "$num" == "Y" ] && echo "OK, continue" && exit

[ "$num" == "n" -o "$num" == "N" ] && echo "Oh, interrupt" && exit

echo " I don't know what is your choice is"

#-o或连接

shell脚本的默认变量

bash 3.sh 1 2 3 4

$0 $1 $2 $3 $4 ....

$# 代表后接的参数个数

$@ 代表 "$1" "$2" "$3" "$4",变都都是独立的(用双引号括起来)

$\* 代表 "$1c$2c$3c$4", 其中c为分隔符,默认空格

#$@与$\*有所不同,记$@即可

#!/bin/bash

echo "script name is $0"

echo "total have $# 参数"

test "$#" -lt "2" && echo "参数太少" && exit

echo "all name is $@"

echo "first is $1"

echo "second is $2"

上面脚本输出 >> (name.sh 1 2 3 4)

script name is 2.sh

total have 4 参数

all name is 1 2 3 4

first is 1

second is 2

shitf 造成参数变量号码偏移

#!/bin/bash

echo "script name is $0"

echo "total have $# 参数"

echo "all name is $@"

shift #偏移,移动变量,默认不填是 1

echo "total have $# 参数"

echo "all name is $@"

shift 3 #偏移3

echo "total have $# 参数"

echo "all name is $@"

上面脚本结果>>

bash 3.sh 1 2 3 4 5 6

script name is 3.sh

total have 6 参数

all name is 1 2 3 4 5 6

total have 5 参数

all name is 2 3 4 5 6

total have 2 参数

all name is 5 6

条件判断

if....then

单层,简单条件判断式

if [ 条件判断式 ]; then

当条件判断成立时,可以进行的命令内容

fi

多层

if [ 条件判断式 ]; then

当条件判断成立时,可以进行的命令内容

else

当条件判断不成立时,可执行的命令

fi

if [ 条件判断式 ]; then

语句

elif [ 条件判断式 ]; then

语句

else

语句

fi

多括号[]之间以 && 或 || 来隔开,括号间的&&或||与命令行执行的状态不同

&& 代表 and

|| 代表 or

例

#!/bin/bash

read -p "please enter (y/n): " yn

if [ "$yn" == "y" ] || [ "$yn" == "Y" ]; then

echo "ok,continue"

exit 0

elif [ "$yn" == "n" ] || [ "$yn" == "N" ]; then

echo "oh, stop"

exit 0

else

echo "i don't know what your choice is"

fi

例 利用$1变量

#!/bin/bash

if [ "$1" == "" ]; then

echo "you need do it like --> ${0} something"

elif [ "$1" == "Hello" ]; then

echo "Hello,how are you"

elif [ "$1" != "Hello" ]; then

echo "you can only input 'Hello'"

else

echo "i don't know what you do"

fi

检测端口打开

#!/bin/bash

#先将netstat -tunlp的结果数据存到内存当中,不用一直执行netstat

testfile="/dev/shm/netstat\_check.txt"

netstat -tunlp > $testfile

testing=$(grep ":80" ${testfile}) #grep :80的结果放到变量中

if [ "$testing" != "" ]; then #有数据说明Port打开,为空说明没有打开

echo "80 port is open"

fi

testing=$(grep ":22" ${testfile})

if [ "$testing" != "" ]; then

echo "22 port is open"

fi

testing=$(grep ":25" ${testfile})

if [ "$testing" != "" ]; then

echo "25 port is open"

fi

testing=$(grep ":21" ${testfile})

if [ "$testing" != "" ]; then

echo "21 port is open"

fi

输入退伍时间,算出剩余退伍天数

#!/bin/bash

read -p "please enter you 退伍 day in: " rday

#!/bin/bash

read -p "please enter you 退伍 day in: " rday

#判断输入格式

ruwu\_day=$(echo $rday | grep "[0-9]\{8\}")

if [ "$ruwu\_day" == "" ]; then

echo "you should input 20200520"

exit 1

fi

#退伍的时间变成总共秒数

declare -i rs=$(date --date="$rday" +%s)

#今天时间的总共秒数

declare -i ts=$(date +%s)

#剩余时间秒数

declare -i sw=$(( ${rs} -${ts} ))

#秒数换算成天数

declare -i day=$(( ${sw}/60/60/24 ))

if [ "$day" -lt "0" ]; then

echo "你已退伍$((-1\*${day}))天"

else

echo "你还剩余 ${day} 天退伍"

fi

#秒数为自19700101累积而来的秒数

case

case $变量名称 in #关键字case,变量前有美刀符号

"第一个变量内容") #每个变量内容用双引号,右括号为关键字

程序段

;; #每个类别结尾用;;处理

"第二个变量内容")

程序段

;;

\*) #最后一个变量内容都用\*, 代表其他值,但不包含上面的变量内容

程序段

;;

esac #结尾反过来写

例，交换式变量

#!/bin/bash

read -p "please input your choicce (1/2/3): " cho

case $cho in

"1")

echo "you choice 1"

;;

"2")

echo "you choice 2"

;;

"3")

echo "you choice 3"

;;

\*)

echo "you should input 1/2/3"

;;

esac

利用$1

#!/bin/bash

case $1 in

"hello")

echo "hello,how are you"

;;

"")

echo "you must enter something like -->${0} someword"

;;

\*)

echo "user ${0} hello"

esac

function(函数)

function fname() {

程序段

}

注意函数一定要放在脚本的最前面,因为脚本是从上面下从左而右执行分析

例

#!/bin/bash

function printit() {

echo -n "your choice is " #-n后续不换行在同一行显示

}

case $1 in

"one")

printit; echo $1 | tr 'a-z' 'A-Z' #大小写转换

;;

"two")

printit; echo $1 | tr 'a-z' 'A-Z'

;;

"three")

printit; echo $1 | tr 'a-z' 'A-Z'

;;

\*)

echo "you should input one/two/three"

;;

esac

function内置变量

#!/bin/bash

function printit() {

echo "your choice is ${1} " $$1与onw无关,要参考下面的命令执行

#echo "your choice is ${1} ${2}"

}

case $1 in

"one")

printit 1 #带参数1

;;

"two")

printit 2

#printit 2 3 #这样会传递两个参数进函数

;;

"three")

printit 3

;;

\*)

echo "you should input one/two/three"

;;

esac

#sh one -->1 sh two-->2 sh three-->3

loop(循环)

while do done, until do done (不固定循环)

while [ conditin ]

do #循环开始

程序段

done #循环结束

#while是当contition成立时,就进行循环

until [ condition ]

do

程序段

donw

#until刚好相反,当条件满足时,终止循环

例 while

#!/bin/bash

while [ "$yn" != "yes" -a "$yn" != "YES" ]

do

read -p "please enter yes: " yn

done

#只要不是输入yes或YES就一直循环让其输入,注意 -a

例until

#!/bin/bash

until [ "$yn" == "yes" -o "$yn" == "YES" ]

do

read -p "plesa enter yes: " yn

done

#只要是yes或YES，就停止输入

例计算1+2+3...+99+100

#!/bin/bash

declare -i sum=0

declare -i total=0

while [ "$sum" -le "100" ]

do

total=$(( $total + $sum ))

sum=$(( $sum + 1))

done

echo "1+2+3+....99+100= $total"

for..do...done(固定循环)

for var in con1 con2 con3....

do

程序段

done

第一次循环时,$var内容为con1

第一次循环时,$var内容为con2

......

例分别输出动物名

#!/bin/bash

for animal in dog cat elephot

do

echo "this is $animal"

done

例用id利用for循环查看账号

#!/bin/bash

for i in $(cat /etc/passwd | cut -d ':' -f 1)

do

id $i

done

~

例用Ping测试主机连线

#!/bin/bash

ip="192.168.1."

host=$(seq 1 20) #sed连续1到20,也可以用{1..20}代替

for i in $host

do

ping -c 1 -w 1 ${ip}${i} &> /dev/null

if [ "$?" == "0" ]; then

echo "${ip}${i} is UP"

else

echo "${ip}${i} is DOWN"

fi

done

例,输入一个目录,用for例出目录内的文件的权限

#!/bin/bash

read -p "please enter a dicrect name: " dic

#判断目录不为空与存在

if [ "${dic}" == "" -o ! -d "$dic" ]; then

echo "you should enter a direct"

exit 1

fi

file=$(ls $dic)

for i in $file

do

test -r ${dic}/${i} && echo "$i have r"

test -w ${dic}/${i} && echo "$i have w"

test -x ${dic}/${i} && echo "$i have x"

done

for..do...done(数值算理)

for (( 初始值;限制值;赋值运算 ))

do

程序段

done

#初始值,某个变量在循环当中的初始值,如i=1

#限制值,当变量的值在这个限制值的范围内,就继续进行循环,如i<=100

#赋值运算,每做一次循环,变量也变化,如i=i+1

例,输入一个数字n然后计算1+2+...n

#!/bin/bash

read -p "please enter a nmuber , i will count 1+2+....number : " nu

if [ "$nu" -le "0" ]; then

echo "you shoul input a number > 0"

exit 1

fi

declare -i sum=0

for (( i=1;$i <= $nu;i++))

do

sum=$(($sum + $i))

done

echo "1+2+....+$nu = $sum"

例随机数与数组结合随机生成吃饭选择

#!/bin/bash

eat[1]="粉之都"

eat[2]="云吞"

eat[3]="生料粉"

eat[4]="外卖"

eat[5]="鸡饭"

eat[6]="鸭饭"

eat[7]="叉烧饭"

eat[8]="猪肉饭"

eat[9]="青菜饭"

rx=9

rand=$((${RANDOM}\*${rx} / 32767 + 1))

echo "today we eat ${eat[$rand]}"

shell脚本的跟踪与调试

-n 不要执行脚本,仅查询语法的问题

如sh -n test.sh若语法没问题，则什么也不会显示

-x 将使用到的脚本内容显示到屏幕上(会执行脚本)

20200602,

linux账号管理与ACL权限设置

linux的账号与用户组

用户的标识符 UID与GID

计算机只会认ID,账号与ID对应只是方便人记忆,id与账号对应在/etc/passwd中

每个登录的用户都会有两个ID，UID(user id)与GID(group id)

文件有 拥有者id与拥有人组id,利用/etc/paswwd与 /etc/group对应显示

如若修改/etc/passwd里的一般账号uid为其它数字,那原先属于该用户的文件的拥有者属性就会变成数字(因为id与user不对应),且该用户下次登录时进不了家目录(权限也跟着变了),所以实际环境中不要随意修改/etc/passwd

用户账号

要登录系统,账入账号与密码后

1 先查找/etc/passwd比对是否有输入的账号,如果没有则退出,如果有就把uid与gid读了来,另外该账号的家目录与shell设置也一并读出

2 然后再核对密码,系统会过入/etc/shadow里面找出对应的账号的Uid然后比对刚刚输入的密码与shadow文件是否一样

3 如果一切顺便就进行shell管理阶段

/etc/passwd文件说明

head -n 3 /etc/passwd

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

每一行用:隔开，共有7段

1 账号名称,用来对应uid

2 密码, 只显示x，为了安全密码数据改放到/etc/shadow

3 uid

0 代表系统管理员,如需其它账号也具有root权限,将账号Uid改为0即可,

不建议多个容易乱

1~999 1000以下留给系统作为保留账号只是一个习惯

1~200 由Linx发行版自行建立的系统账号

201~999 若用户有系统账号需求，可以用

1000~60000 给一般用户使用

4 GID 与/etc/group有关,规范组名与GID对应而已

5 用户信息栏说明

6 家目录,可以修改

7 shell

/etc/shadow 文件结构

head -n 3 /etc/shadow

root:$6$esu2az6WJraoEBnx$4qVMw/P4z4gn2S/RFhDD/SiyY2MZsM6xlfro8EFxrcvldSweyqFcF.OWrm2/vUG/.VXc5a4R2yDioqr1NS3Aw/::0:99999:7:::

bin:\*:17834:0:99999:7:::

1 账号名称

2 密码(经过编码的密码(摘要)),目前常用SHA编码技术,一些软件会在可在此段前加上！或\*让它(暂时失效)

3 最近修改密码的日期,如bin17834则是以1970年1月1日到现在的天数

4 密码不可被修改的天数(与3相比),记录最近一次被更改后需要经过几天才可以再被修改

5 密码需要重新修改的天数(与3相比),记录最近一次更改密码后,在几天内需要再次修改密码,如上的99999就没有强制的意思

6 密码需要修改期限前的警告天数(与5相比),当密码的有效期快要到时(5),系统会根据这个字段的设置发了警告信息给这个账号,如上面的7,密码到期前7天之内,系统会警告该用户

7 密码过期后的账号宽限时间(密码失效日)与(5相比),密码的有效日期为3(更新日期)+5(重新修改日期),过了该期限后用户依旧没有更改密码,那密码就算过期,过期后该账号还是可以登陆系统获得shell,但登录后系统会强制要求你必须重新设置密码才能继续使用,该字段的功能就是密码过期几天后,用户还是没有改密码,那这个账号的密码将会失效,即该账号再也无法使用该密码登录

8 账号失效时间

9 保留

密码忘记

一般用户密码忘记可以用root通过passwd命令直接重置密码即可

root密码忘记

重启进入单人维护模式,系统会主动给予root权限的bash接口，再以passwd修改

以live cd启动后挂载根目录云修改/etc/shadow，将root的密码字段清空,重启空再修改密码

用户组

/etc/group

head -n 2 /etc/group

root:x:0:

bin:x:1:

1 组名,3 GID 对应

2 用户组密码(很少用)

3 GID, /etc/passwd第四个字段使用的GID对应的用户组名,就是这里对应出来

4 此用户组支持的账号名称,一个账号可以加入多个用户组,某账号想加入此组,将账号填入这字段即可(新版的linux中,初始用户组的用户群已经不会加入这个字段)

有效用户组与初始用户组

/etc/passwd的第4栏的GID,就是所谓的初始用户组,用户一登录系统就会马上拥有这个用户组的相关权限

[root@localhost ~]# grep dmtsai /etc/passwd /etc/group /etc/gshadow

/etc/passwd:dmtsai:x:1003:1004::/home/dmtsai:/bin/bash

/etc/group:users:x:100:dmtsai

/etc/group:dmtsai:x:1004:

/etc/gshadow:users:::dmtsai

/etc/gshadow:dmtsai:!::

#在passwd里看到,dmtsai的GID为1004,/etc/group对应下的1004为dmtsai,这就是初始用户组,所以并不需要在第4字段写入该账号

而users并非dmtsai的初始用户组,所以要第第4栏加入,这样dmtsai才够加入users这个组 #usermod -a -G users dmtsai

因此dmtsai账号同时支持dmtsai与users这两个用户组,在rwx文件时,针对用户组部分，只要是users与dmtsai拥用的功能,dmtsai用户都拥有

新文件的用户组是哪个还要看有效用户组(effctive group)

[dmtsai@localhost ~]$ groups #查看有效与支持的用户组

dmtsai users

#支持两个组，有效用户组为第一个dmtsai

touch test

[dmtsai@localhost ~]$ ll

total 0

-rw-rw-r--. 1 dmtsai dmtsai 0 Jun 2 21:26 test

#有效用户组新建文件的属性都是dmtsai

修改有效用户组,前提是想要切换的用户组必须是已经有支持的用户组

[dmtsai@localhost ~]$ newgrp users

[dmtsai@localhost ~]$ groups

users dmtsai

[dmtsai@localhost ~]$ touch test2

[dmtsai@localhost ~]$ ll test2

-rw-r--r--. 1 dmtsai users 0 Jun 2 21:35 test2

[dmtsai@localhost ~]$ groups

users dmtsai

[dmtsai@localhost ~]$ exit

exit

[dmtsai@localhost ~]$ groups

dmtsai users

#注意newgrp可以修改用户的有效用户组,而且是另外以一个shell来提供这个功能的,因此要回到原来的环境中要exit

用户加入不同的用户组有两个方法,一个是root利用usermod，另一个是通过用户组管理员以gshadow帮加入

[root@localhost ~]# head -n 3 /etc/gshadow

root:::

bin:::

daemon:::

1 组名

2 密码, 如为！或 空表示无合法密码,所以无用户组管理员

3 用户组管理员的账号

4 有加入该用户组的账号(与/etc/group相同)

gshadow最大的功能就是建立用户组管理员,不过现在有sudo，这功能少用

账号管理

新增用户

useradd [-u UID] [-g 初始用户组] [ -G 次要用户组] [-mM] \

[-c 说明栏] [-d 家目录绝对路径] [-s shell] 账号

-u uid,直接指定一个特定的UID给账号

-g 接的是初始用户组,这个用户组的GID会被放到/etc/passwd的第4栏位

-G 后面接的用户组则是该账号还可加入的用户组

-M 强制,不要建立家目录(系统账号默认值)

-m 强制,要建立使用者家目录(一般账号默认值)

-c /etc/passwd第5栏,随便设置

-d 指定某个目录成为家目录,需不使用默认值，要使用绝对路径

-r 建立一个系统账号

-s 后面接一个shell，默认/bin/bash

-e 后面接一个日期,格式YYYY-MM-DD,账号失效日期

-f shadow第7栏选项,指定密码是否会失效,0为立刻失效

-l 永不失效

例用默认值建立一个账号

[root@localhost ~]# useradd virbd1

[root@localhost ~]# ll -d /home/virbd1/

drwx------. 2 virbd1 virbd1 62 Jun 2 22:25 /home/virbd1/

[root@localhost ~]# grep virbd1 /etc/passwd /etc/shadow /etc/group

/etc/passwd:virbd1:x:1004:1005::/home/virbd1:/bin/bash

/etc/shadow:virbd1:!!:18415:0:99999:7:::

/etc/group:virbd1:x:1005:

#注意会在group里面加入一个与账号名称一样的组名,会在/home下面建立一个与账号同名的目录作为家目录，权限为700

例已知有个users用户组,且UID1500不存在,请用users为初始用户组,1500为UID建立一个vbird2的账号

useradd -u 1500 -g users vbird2

#注意,因为是指定一个存在的用户组做为初始用户组,所以就不会在group里主动建立与账号同名的用户组了

例建立系统账号

useradd -r vbird3

[root@localhost ~]# ll -d /home/vbird3

ls: cannot access /home/vbird3: No such file or directory

[root@localhost ~]# grep vbird3 /etc/passwd /etc/shadow /etc/group

/etc/passwd:vbird3:x:996:993::/home/vbird3:/bin/bash

/etc/shadow:vbird3:!!:18415::::::

/etc/group:vbird3:x:993:

#系统账号uid一般应该小于1000,且默认主动不会建立家目录

useradd参考文件

root@localhost skel]# useradd -D #默认值参考

GROUP=100

HOME=/home

INACTIVE=-1

EXPIRE=

SHELL=/bin/bash

SKEL=/etc/skel #家目录的内容数据参考目录,也可以增加文件

CREATE\_MAIL\_SPOOL=yes #建立用户的mailbox,如/var/spool/mail/vbird1

注意GROUP=100,针对用户组角度有两种机制

私有用户组机制,系统会建立一个与账号同名的用户组给用户作为初始用户组,比较安全(centos使用私有用户组等)

公共用户组机制,每个账号都属于user这个用户组(GID)

uid与gid参考文件

root@localhost skel]# cat /etc/login.defs | grep -v "^$" | grep -v '^#'

MAIL\_DIR /var/spool/mail

PASS\_MAX\_DAYS 99999

PASS\_MIN\_DAYS 0

PASS\_MIN\_LEN 5

PASS\_WARN\_AGE 7

UID\_MIN 1000

UID\_MAX 60000

SYS\_UID\_MIN 201

SYS\_UID\_MAX 999

GID\_MIN 1000

GID\_MAX 60000

SYS\_GID\_MIN 201

SYS\_GID\_MAX 999

CREATE\_HOME yes

UMASK 077

USERGROUPS\_ENAB yes

ENCRYPT\_METHOD SHA512

#系统设置一个账号uid时,会先是参考uid\_min最小值后再找出passwd最大uid数值,两者相比找出最大的再加1就是新账号的UID

USERGROUPS\_ENAB的功能是如果使用userdel删除一个账号,且该账号所属的初始用户组已经没有人隶属于该用户组了,那么就删除掉该用户组

passwd

修改密码

--stdin 可以通过来自前一个管道的数据,作为密码输入,对shell脚本有帮助

-l lock的意思,会将/etc/shadow第二栏最前面加上!使密码失效

-u unlock

-S 列出密码相关参数

-n 后接天数,shadow第4栏位,多久不可修改密码

-x 后接天数,shadow第5栏位,多久内必须修改密码

-w 后接天数,shadow第6栏位,密码过期前的警告天数

-i 后接天数,shadow第7栏位,密码失效日期

passwd 账号 #修改一般账号密码,后不接账号为修改自己的密码

密码不能与账号相同

密码尽量不要选用字典里面出现的字符串

密码必须超过8个字符

密码尽量使用大小写符,数字,特殊字符$ - 等组合

例用--stdin建立密码

echo "abc123456" | passwd --stdin vbird2

#缺点会在命令历史中出现,通常配合shell脚本

passwd -S vbird2

vbird2 PS 2020-06-02 0 99999 7 -1 (Password set, SHA512 crypt.)

#密码建立时间20200602,99999修改天数,7警告日数与密码不会失效-1

例管理vbird2密码使具有60天修改,密码过期10天后账号失效

[root@localhost pam.d]# passwd -x 60 -i 10 vbird2

Adjusting aging data for user vbird2.

passwd: Success

[root@localhost pam.d]# passwd -S vbird2

vbird2 PS 2020-06-02 0 60 7 10 (Password set, SHA512 crypt.)

锁住用户

[root@localhost pam.d]# passwd -l vbird2

Locking password for user vbird2.

passwd: Success

[root@localhost pam.d]# passwd -S vbird2

vbird2 LK 2020-06-02 0 60 7 10 (Password locked.)

[root@localhost pam.d]# grep 'vbird2' /etc/shadow

vbird2:!!$6$.CAwZp2y$fltb7soZ6a2gY29ARCBleF7KjToV9GGrcEqX8n7.ia2dH8L.oZLRXwtFBFxYHUG5mgQCgr8IVySQyA6oresEn1:18415:0:60:7:10::

[root@localhost pam.d]# passwd -u vbird2 #解锁

Unlocking password for user vbird2.

passwd: Success

#锁住用户其实是在shadow第二栏加了两个！,

chage

chage -l vbird2

Last password change : Jun 02, 2020

Password expires : Aug 01, 2020

Password inactive : Aug 11, 2020

Account expires : never

Minimum number of days between password change : 0

Maximum number of days between password change : 60

Number of days of warning before password expires : 7

chage可以更详细的显示密码参数,也可以修改设置值

例用户一登录就强制它们一定要修改密码后才能使用

[root@localhost pam.d]# useradd agetest

[root@localhost pam.d]# echo "agetest" | passwd --stdin agetest

Changing password for user agetest.

passwd: all authentication tokens updated successfully.

[root@localhost pam.d]# chage -d 0 agetest

[root@localhost pam.d]# chage -l agetest

Last password change : password must be changed

Password expires : password must be changed

Password inactive : password must be changed

usermod

账号参数微调

让账号失效

usermod -e "2020-6-30" vbird2

[root@localhost ~]# chage -l vbird2

Last password change : Jun 02, 2020

Password expires : Aug 01, 2020

Password inactive : Aug 11, 2020

Account expires : Jun 30, 2020

为vbird3创建家目录

[root@localhost ~]# ll -d /home/vbird3

ls: cannot access /home/vbird3: No such file or directory

[root@localhost ~]# cp -a /etc/skel /home/vbird3

[root@localhost ~]# chown -R vbird3:vbird3 /home/vbird3

[root@localhost ~]# chmod 700 /home/vbird3

#chown -R 连同家目录的使用者/用户组属性都一起修改

#chmod没有-R,因为仅修改目录的权限而非内部文件的权限

userdel

删除用户

#连同家目录一起删除

userdel -r vbird2

userdel: user vbird2 is currently used by process 15027

#注意如果账户只是暂时不用,可以将passwd的账号失效日期(第8字段)为0即可,确认账号真的不需要再删除,可以用find / -user username来查看整个系统属于username的文件

用户功能

id

uid=0(root) gid=0(root) groups=0(root) context=unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023

#可以查看uidgid与支持的用户组,context是selinux先不理

id vbird100

id: vbird100: no such user

#也可以利用id判断用户是否存在

finger

查看用户相关信息

chfn

change finger

chsh

change shell

#以上三个少用

新增与删除用户组

group

-g 后接特定GID,指定直接设置GID

-n 建立系统用户组

groupadd group1

groupmod

-g 修改即有GID

-n 修改即有的用户组名称

修改group1的名称为mygroup，GID为201

groupmod -g 201 -n mygroup group1

#不要随意改GID,造成资源错乱

groupdel

删除用户组

groupdel mygroub

groupdel vbird1

groupdel: cannot remove the primary group fo user 'vbird1'

#不能删除,原因某个账号的初始用户组使用该用户组

删除方法

修改vbird1 GID

删除vbird1这个用户

用户组管理员功能

gpasswd

关于系统管理员(root)做的操作

gpasswd groupname

gpasswd [-A user1,...] [-M user3,...] groupname

gpasswd [-rR] groupname

: 若没有任何参数,表示设置groupname密码

-A 将groupname的管理权交由后面的使用者管理

-M 将某些账号加入这个用户组

-r 将groupname的密码删除

-R 让groupname的密码失效

关于用户组管理员做的操作

-a 将某位使用者加入到groupname这个用户组中

-d 将某位使用者删除出用户组

groupadd testgroup #添加一个用户组

gpasswd testgroup #设置用户组密码

[root@localhost ~]# gpasswd -A vbird1 testgroup #加入用户组管理员为vbird1

[root@localhost ~]# grep testgroup /etc/group /etc/gshadow

/etc/group:testgroup:x:1502:

/etc/gshadow:testgroup:$6$atXSm.BoB$XmXx4ZF1jhMjVTobmBHo18alp4r2vtkULI45iHDuya7bPcLicXd6Qx.bwV9UYcGHAmnsJxy/xX7OBppPi82Xf1:vbird1:

[root@localhost ~]# su - vbird1

[vbird1@localhost ~]$ id vbird1

uid=1502(vbird1) gid=1503(vbird1) groups=1503(vbird1)

[vbird1@localhost ~]$ gpasswd -a vbird1 testgroup #用户组管理员给组加入成员

Adding user vbird1 to group testgroup

[vbird1@localhost ~]$ id

uid=1502(vbird1) gid=1503(vbird1) groups=1503(vbird1) context=unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023

[vbird1@localhost ~]$ gpasswd -a vbird3 testgroup #用户组管理员给组加入成员

Adding user vbird3 to group testgroup

[vbird1@localhost ~]$ grep testgroup /etc/group

testgroup:x:1502:vbird1,vbird3

例用户pro1 pro2 pro3为同一个项目计划的开发人员,让这三个用户在同一个目录下工作,但这三个目录还是拥有自己的家目录与基本的私用户组,工作目录为/srv/projecta

[root@localhost ~]# mkdir /srv/projecta

[root@localhost ~]# groupadd progroup #建立公共用户组

[root@localhost ~]# chown root:progroup -R /srv/projecta/

[root@localhost ~]# ll -d /srv/projecta/

drwxr-xr-x. 2 root progroup 6 Jun 3 14:38 /srv/projecta/

[root@localhost ~]# useradd -G progroup pro1 #让账号支持用户组

[root@localhost ~]# useradd -G progroup pro2 root@localhost ~]# useradd -G progroup pro3

[root@localhost ~]# grep 'progroup' /etc/group

progroup:x:1508:pro1,pro2,pro3

[root@localhost ~]# chmod 2770 /srv/projecta/ #设置权限(包括SGID,这样用户新建的文件用户组属于公共用户组)

[root@localhost ~]# ll -d /srv/projecta/

drwxrws---. 2 root progroup 6 Jun 3 14:38 /srv/projecta/

账号设置密码省略

#以上例子不能针对某个特定账户设置专属权限,如myuser3我只想让其能查看不能修改数据,让其不加入公共组给目录其它权限rx，但这样其它用户也可以访问数据

主机的详细权限规划 ACL

传统的linux权限只能会对一个用户,一个用户组,非此用户组的三种身份权限设置

ACL可以针对单一用户,用户组,单一文件或目录等来设置权限

ACL已经默认加入了所有常见的的Linux文件系统的挂载参数中

查看是否支持ACL

dmesg | grep -i acl #有数据即可

setfacl #设置某个目录或文件的ACL

-m 设置后续的ACL参数给文件使用,不可与-x合用

-x 删除后续的ACL参数,不可与-m合用

-b 删除所有的ACL设置参数

-k 删除(默认的)ACL参数

-R 递归设置ACL,即包括子目录都会被设置起来

-d 设置默认acl参数,只对目录有效,在该目录新建的数据会引用止默认值

例针对特定使用者的方式

touch acl\_test1

[root@localhost ~]# ll acl\_test1

-rw-r--r--. 1 root root 0 Jun 3 16:34 acl\_test1

[root@localhost ~]# setfacl -m u:vbird1:rx acl\_test1

[root@localhost ~]# ll acl\_test1

-rw-r-xr--+ 1 root root 0 Jun 3 16:34 acl\_test1 #注意后面有个+,且与原本的644权限有差异

[root@localhost ~]# setfacl -m u::rwx acl\_test1 #无用户说明是设置该文件拥有者

[root@localhost ~]# ll acl\_test1

-rwxr-xr--+ 1 root root 0 Jun 3 16:34 acl\_test

查看ACL

getfacl acl\_test1

# file: acl\_test1 #文件明

# owner: root #文件拥有者

# group: root #文件所属用户组

user::rwx #使用者列表为空代表文件拥有者的权限

user:vbird1:r-x #针对vbird1的权限为rx

group::r-- #针对文件用户组的权限设置为r

mask::r-x #此文件默认有效权限(mask)

other::r-- #其他人权限

针对特定用户组的设置

setfacl -m g:mygroup1:rx acl\_test1

[root@localhost ~]# getfacl acl\_test1

# file: acl\_test1

# owner: root

# group: root

user::rwx

user:vbird1:r-x

group::r--

group:mygroup1:r-x #新增

mask::r-x

针对有效权限设置

setfacl -m m:r acl\_test1

[root@localhost ~]# getfacl acl\_test1

# file: acl\_test1

# owner: root

# group: root

user::rwx

user:vbird1:r-x #effective:r--

group::r--

group:mygroup1:r-x #effective:r--

mask::r--

other::r--

#mask代表用户或用户组所设置的权限必须要存在于mask的权限设置范围内才会生效,上面的例子mask仅为r，所以vbird1与mygroup虽有设置rx,但却只有r权限,通常mask设置rwx

承接pro项目例子,让myuser1只能查看目录数据,不能修改

setfacl -m u:myuser1:rx /srv/projecta #设置rx权限

[root@localhost ~]# su - myuser1

Last login: Wed Jun 3 17:04:33 CST 2020 on pts/0

[myuser1@localhost ~]$ cd /srv/projecta/

[myuser1@localhost projecta]$ ll

total 0

[myuser1@localhost projecta]$ ls -la #确实可以查询数据(也可以复制到其它目录)

total 0

drwxrws---+ 2 root progroup 6 Jun 3 14:38 .

drwxr-xr-x. 5 root root 52 Jun 3 14:38 ..

[myuser1@localhost projecta]$ touch test #不能修改

touch: cannot touch ‘test’: Permission denied

例新建文件查看属性

[root@localhost projecta]# ll

drwxr-sr-x. 2 root progroup 18 Jun 3 17:19 abc

-rw-r--r--. 1 root progroup 6 Jun 3 17:07 test

#发现文件后面并没有+号,代表acl属性没有继承,其它人属性还有

使用默认权限设置让acl在目录下面的数据都有继承功能

setfacl -m d:u:myuser1:rx /srv/projecta

[root@localhost projecta]# ll

total 0

drwxrws---+ 2 root progroup 6 Jun 3 21:48 abc

-rw-rw----+ 1 root progroup 0 Jun 3 21:40 zzz1

drwxrws---+ 2 root progroup 6 Jun 3 21:40 zzz2

[root@localhost projecta]# getfacl abc

# file: abc

# owner: root

# group: progroup

# flags: -s-

user::rwx

user:myuser1:r-x

group::rwx

mask::rwx

other::---

default:user::rwx

default:user:myuser1:r-x

default:group::rwx

default:mask::rwx

default:other::---

#default也有了，继承成功,myuser1继续拥有子目录与文件的acl权限

例,承接上面,取消myuser1的设置,让pro3这个用户无法使用该目录

setfacl -x u:myuser1 /srv/projecta #取消myuser1的ACL设置

setfacl -x d:myuser1 /srv/projecta #取消myuser1的ACL默认值设置

setfacl -m u:pro3:- /srv/projecta

#设置一个用户或用户组没有权限时不能留空白,用-

[myuser1@localhost ~]$ getfacl /srv/projecta/

getfacl: Removing leading '/' from absolute path names

# file: srv/projecta/

# owner: root

# group: progroup

# flags: -s-

user::rwx

user:pro3:--- #pro3没有任何权限

用户身份切换

一般情况下用一般用户操作,需要用户root权限时再切换

像apache这些程序可以建立一个名为apache的用户来启动apache,相对安全,程序被攻击,不至于系统会损坏

su

切换身份

- 单纯使用su - 代表使用login-shell的变量文件读取方式来登录系统,后无账号则是切换为root

-l 与- 类似,也是login-shell登录,但后面要接账号

-c 仅进行一次命令,所以-c后面可接命令

一般用户切换为root,要用

su -

root切换为其它等用

su -l 账号

su - -c "head -n 3 /etc/shadow" #仅用root执行一次命令

su缺点是一般用户都要知道root的密码,不安全

sudo

一开始系统默认的只有root可以执行

-b 将后续的命令放到后台中让系统自行执行,不与目前的shell产生影响

-u 后面可以接欲切换的使用者,若无此项则代表切换身份为root

例以sshd身份在/tmp下建立一个mysshd文件

sudo -u sshd touch /tmp/mysshd #用root创建然后再chown也行

以vbird1身份建立~vbird1/www并于其中建立index.html文件

sudo -u vbird1 sh -c "mkdir ~vbird1/www;cd ~vbird1/www; \

echo 'This is a test'>index.html"

#~vbird1就是概用户根目录

visudo

能否执行sudo与/etc/sudoers设置有关,不建议用vi去编辑/etc/sudoers,使用visudo，退出visudo退其会自行检测语法正确性

单一用户可使用root所有命令

visudo

....

root ALL=(ALL) ALL

vbird1 ALL=(ALL) ALL

#各字段意思

1 root 操作系统的哪个账号可以使用sudo

2 ALL 登录者来源主机名,设置可以指定客户端计算机(别的机子来源),默认值root可来自任何一台主机

3 ALL 可切换的身份 这个账号可以切换成什么身份来执行后续的命令,默认root可以切换任何人

4 ALL 可执行的命令,可用该身份执行什么命令,命令要使用绝对路径,默认root可以切换任何身份且进行任何命令

上面vbird1加入sudo了

例

[vbird1@localhost ~]$ sudo tail -n 1 /etc/shadow

#再输入密码

利用wheel用户组以及免密码的功能处理visudo

visudo

....

%wheel ALL=(ALL) ALL

# % 在最左边加上%代表后面接的是一个用户组

usermod -a -G wheel pro1 #让pro1加入wheel组，这样pro1就拥有sudo权限了

sudo免密

# %wheel ALL=(ALL) NOPASSWD: ALL

#把#去掉利用此条设置

有限制的命令操作

myuser1 ALL=(root) !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]\*, !/usr/bin/passwd root

#给myuser1拥用给其它用户修改密码的权限,且禁止修改root密码

别名设置visudo

User\_Alias ADMPW = pro1, pro2, pro3

Cmnd\_Alias ADMPWCOM = !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]\*, !/usr/bin/passwd root

ADMPW ALL=(root) ADMPWCOM

#ADMPW为创建的用户(要大写),后接实际用户，AMDPWCOM为命令别名(大写)

sudo搭配su

例让用户不用输入root密码而是输入自己的密码切换成root

User\_Alias ADMPW = pro1, pro2, pro3

ADMPW ALL=(root) /bin/su -

PAM模块(插入式验证模块)

一个API,它提供一连串的验证,只要用户将验主阶段的需求告知PAM后,PAM就能够返回用户验证的结果(成功或失败),它是一套验证机制,可以给其它程序调用

如passwd调用PAM的流程

用户开始执行/usr/bin/passwd这个程序,并输入密码

PAM模块会到/etc/pam.d找寻与程序(passwd)同名的配置文件

将验证结果(成功,失败,其他信息)返回给passwd程序

passwd程序会根据PAM返回的结果决定下一个操作(重新输入新密码或通过)

cat /etc/pam.d/passwd

#%PAM-1.0 #版本说明

auth include system-auth #每一行都是一个验证的过程

account include system-auth

password substack system-auth

-password optional pam\_gnome\_keyring.so use\_authtok

password substack postlogin

验证类别 控制标准 PAM模块与该模块的参数

#include是包含引用后面的system-auth设置

PAM的验证机制流程

验证阶段(auth)

授权阶段(account)

密码阶段(password)

会话阶段(session)

详细参考鸟哥Linux13章账号管理

其它文件

limits.conf

与PAM相关的文件,ulimit功能系统管理员可以通过PAM管理,ulimit命令执行只是针对当前的SHELL生效,limits.conf可以针对用户或用户组

例设置vbird1只能建立100M的文件，且大于9000会警告

#@student - maxlogins 4

vbird1 soft fsize 9000

vird1 hard fsize 10000

#第一栏用户或用户组(前面要加@),第4栏为限制的值,单位kb

测试

[vbird1@localhost ~]$ dd if=/dev/zero of=test bs=1M count=110

File size limit exceeded

例限制用户组，每次仅能有一个使用者登录系统（maxlogins)

@pro1 hard maxlogins 1

#注意只对初始用户组有效

limit.conf修改完成后不用重启任何服务就生效,但是对已登录的用户无效,要下一次登录才生效,因为PAM是程序调用时才设置的

/var/log/secure /var/log/message

如发生无法登录或其它错误时,可以查看这两个日志文件

主机上的用户信息传递

查询用户

[root@localhost ~]# w #与who差不多

14:28:35 up 8 days, 7:39, 3 users, load average: 0.00, 0.01, 0.05

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

root tty1 22May20 6days 0.71s 0.71s -bash

root pts/0 192.168.1.7 05Jun20 6days 0.49s 0.12s -bash

root pts/1 192.168.1.7 05Jun20 3.00s 0.08s 0.00s w

[root@localhost ~]# lastlog #与last相似

Username Port From Latest

root pts/1 192.168.1.7 Fri Jun 5 11:10:57 +0800 2020

bin \*\*Never logged in\*\*

daemon \*\*Never logged in\*\*

adm \*\*Never logged in\*\*

#可查看所有用户(包括系统用户)登录时间

用户对谈

write

针对一个用户来发送信息

write root tty1

hello

.... #control + d完成发送的信息

root用户可以mesg 拒绝收一其它用户的信息,但一般用户不能拒绝root发的信息

mesg n

mesg #查看mesg是y或n

wall

对所有系统上面的用户发送信息

wall "hello, i will shutdown server..."

#用"字符发送消息",回车即发送完成

mail

wall write要等到用户在线才能收信息

发送邮件

mail -s "邮件标题" username@localhost

寄给本机用户不用@localhost

mail -s "hello" vbird1

hello

nice to meet you

you

. #结束标志

EOT

利用数据流重定向写入内容

mail -s "bashrc file content" vbird1 < ~/.bashrc

通过管道将ls -al ~内容传给邮件

ls -al ~ | mail -s "myfile" vbird1

收邮件

[root@localhost ~]# mail

Heirloom Mail version 12.5 7/5/10. Type ? for help.

"/var/spool/mail/root": 2 messages

> 1 leison@localhost.loc Wed Jun 3 22:56 17/698 "\*\*\* SECURITY information for "

2 root Thu Jun 11 20:00 63/3163 "myfile"

&

# & 为提示符

? 可获得帮助

常用命令

h 列出邮件标头,如要查看40封邮件左右的邮件标头 h 40

d 删除邮件, 如删除第10封 d10 , 删除第20~40，d20-40

s 将邮件保存为文件, 如保存第5封, s 5 ~/mail.file

x 不做任何操作退出,无论刚刚是删除还是阅读x后都会无效

q 保存后退出

（插入学习 iptables)

iptables是按顺序比对规则(从上而下),若封包进入比对rule1符合时,就会对这个封包进行相应的action 1动作,而不会理会后续的rule2,rule3....,若不符合rule1,就会比对rule2,rule3...,如果所有的规则都不符合,就会透过预设动作来决定封包的去向

例提供www服务的服务器,发现192.168.100.100恶意尝试入侵系统,拒绝它

1 rule1 先抵挡192.168.100.100

2 rule2 再让要求www服务的封包通过

3 rule3 将所有封包丢弃

以上为正确的，若如下

1 rule2 再让要求www服务的封包通过

2 rule1 先抵挡192.168.100.100

3 rule3 将所有封包丢弃

这样192IP就已经让它通过了而不会比对第2条规则

Linux的iptables至少就有三个表格，包括管理本机进出的filter 、管理后端主机(防火墙内部的其他电脑)的nat 、管理特殊旗标使用的mangle (较少使用)

每个表格与其中链的用途分别是这样的：

filter (过滤器)：主要跟进入Linux本机的封包有关，这个是预设的table喔！

INPUT：主要与想要进入我们Linux本机的封包有关；

OUTPUT：主要与我们Linux本机所要送出的封包有关；

FORWARD：这个咚咚与Linux本机比较没有关系，他可以『转递封包』到后端的电脑中，与下列nat table相关性较高。

nat (位址转换)：是Network Address Translation的缩写，这个表格主要在进行来源与目的之IP或port的转换，与Linux本机较无关，主要与Linux主机后的区域网路内电脑较有相关。

PREROUTING：在进行路由判断之前所要进行的规则(DNAT/REDIRECT)

POSTROUTING：在进行路由判断之后所要进行的规则(SNAT/MASQUERADE)

OUTPUT：与发送出去的封包有关

mangle (破坏者)：这个表格主要是与特殊的封包的路由旗标有关，早期仅有PREROUTING及OUTPUT链，不过从kernel 2.4.18之后加入了INPUT及FORWARD链。由于这个表格与特殊旗标相关性较高，所以像咱们这种单纯的环境当中，较少使用mangle这个表格。

观察防火墙

[root@www ~]# iptables-save [-t table]

选项与参数：

-t ：可以仅针对某些表格来输出，例如仅针对nat 或filter 等等

[root@www ~]# iptables-save

# Generated by iptables-save v1.4.7 on Fri Jul 22 15:51:52 2011

\*filter <==星号开头的指的是表格，这里为filter

:INPUT ACCEPT [0:0] <==冒号开头的指的是链，三条内建的链

:FORWARD ACCEPT [0:0] <= =三条内建链的政策都是ACCEPT啰！

:OUTPUT ACCEPT [680:100461]

-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT <==针对INPUT的规则

-A INPUT -p icmp -j ACCEPT

-A INPUT -i lo -j ACCEPT <==这条很重要！针对本机内部介面开放！

-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT

-A INPUT -j REJECT --reject-with icmp-host-prohibited

-A FORWARD -j REJECT --reject-with icmp-host-prohibited <==针对FORWARD的规则

COMMIT

# Completed on Fri Jul 22 15:51:52 2011

清除规则

[root@www ~]# iptables [-t tables] [-FXZ]

选项与参数：

-F ：清除所有的已订定的规则；

-X ：杀掉所有使用者"自订" 的chain (应该说的是tables ）啰；

-Z ：将所有的chain 的计数与流量统计都归零

范例：清除本机防火墙(filter)的所有规则

[root@www ~]# iptables -F

[root@www ~]# iptables -X

[root@www ~]# iptables -Z

预定规则,当封包不在规则设定之内时,则封包的通过与否,以预定规则(policy)为准

常都是将INPUT的policy定义为DROP啦，其他两个则定义为ACCEPT

[root@www ~]# iptables [-t nat] -P [INPUT,OUTPUT,FORWARD] [ACCEPT,DROP]

选项与参数：

-P ：定义政策( Policy )。注意，这个P 为大写啊！

ACCEPT ：该封包可接受

DROP ：该封包直接丢弃，不会让client 端知道为何被丢弃。

范例：将本机的INPUT设定为DROP ，其他设定为ACCEPT

[root@www ~]# iptables -P INPUT DROP

[root@www ~]# iptables -P OUTPUT ACCEPT

[root@www ~]# iptables -P FORWARD ACCEPT

[root@www ~]# iptables-save

# Generated by iptables-save v1.4.7 on Fri Jul 22 15:56:34 2011

\*filter

:INPUT DROP [0:0]

:FORWARD ACCEPT [0:0]

:OUTPUT ACCEPT [0:0]

COMMIT

# Completed on Fri Jul 22 15:56:34 2011

# 由于INPUT 设定为DROP 而又尚未有任何规则，所以上面的输出结果显示：

# 所有的封包都无法进入你的主机！是不通的防火墙设定！(网路连线是双向的)

基础比对语法

[root@www ~]# iptables [-AI链名] [-io网路介面] [-p协定] \

> [-s来源IP/网域] [-d目标IP/网域] -j [ACCEPT |DROP|REJECT|LOG]

选项与参数：

-AI 链名：针对某的链进行规则的"插入" 或"累加"

-A ：新增加一条规则，该规则增加在原本规则的最后面。例如原本已经有四条规则，

使用-A 就可以加上第五条规则！

-I ：插入一条规则。如果没有指定此规则的顺序，预设是插入变成第一条规则。

例如原本有四条规则，使用-I 则该规则变成第一条，而原本四条变成2~5 号

链：有INPUT, OUTPUT, FORWARD 等，此链名称又与-io 有关，请看底下。

-io 网路介面：设定封包进出的介面规范

-i ：封包所进入的那个网路介面，例如eth0, lo 等介面。需与INPUT 链配合；

-o ：封包所传出的那个网路介面，需与OUTPUT 链配合；

-p 协定：设定此规则适用于哪种封包格式

主要的封包格式有： tcp, udp, icmp 及all 。

-s 来源IP/网域：设定此规则之封包的来源项目，可指定单纯的IP 或包括网域，例如：

IP ：192.168.0.100

网域：192.168.0.0/24, 192.168.0.0/255.255.255.0 均可。

若规范为『不许』时，则加上! 即可，例如：

-s ! 192.168.100.0/24 表示不许192.168.100.0/24 之封包来源；

-d 目标IP/网域：同-s ，只不过这里指的是目标的IP 或网域。

-j ：后面接动作，主要的动作有接受(ACCEPT)、丢弃(DROP)、拒绝(REJECT)及记录(LOG)

范例：设定lo成为受信任的装置，亦即进出lo的封包都予以接受

[root@www ~]# iptables -A INPUT -i lo -j ACCEPT

没有列出-s, -d等等的规则，这表示：不论封包来自何处或去到哪里，只要是来自lo这个介面，就予以接受！这个观念挺重要的，就是『没有指定的项目，则表示该项目完全接受

范例：只要是来自内网的(192.168.100.0/24)的封包通通接受

[root@www ~]# iptables -A INPUT -i eth1 -s 192.168.100.0/24 -j ACCEPT

#由于是内网就接受，因此也可以称之为『信任网域』啰。

范例：只要是来自192.168.100.10就接受，但192.168.100.230这个恶意来源就丢弃

[root@www ~]# iptables -A INPUT -i eth1 -s 192.168.100.10 -j ACCEPT

[root@www ~]# iptables -A INPUT -i eth1 -s 192.168.100.230 -j DROP

#针对单一IP来源，可视为信任主机或者是不信任的恶意来源喔！

记录某条规则的纪录

iptables -A INPUT -s 192.168.2.200 -j LOG

只要有封包来自192.168.2.200这个IP时，那么该封包的相关资讯就会被写入到核心讯息，亦即是/var/log/messages这个档案当中。 然后该封包会继续进行后续的规则比对。所以说， LOG这个动作仅在进行记录而已，并不会影响到这个封包的其他规则比对的

TCP UDP端口比对

[root@www ~]# iptables [-AI链] [-io网路介面] [-p tcp,udp] \

> [-s来源IP/网域] [--sport埠口范围] \

> [- d目标IP/网域] [--dport埠口范围] -j [ACCEPT|DROP|REJECT]

选项与参数：

--sport 埠口范围：限制来源的埠口号码，埠口号码可以是连续的，例如1024:65535

--dport 埠口范围：限制目标的埠口号码。

范例：想要连线进入本机port 21的封包都抵挡掉：

[root@www ~]# iptables -A INPUT -i eth0 -p tcp --dport 21 -j DROP

范例：想连到我这部主机的网芳(upd port 137,138 tcp port 139,445)就放行

[root@www ~]# iptables -A INPUT -i eth0 -p udp --dport 137:138 -j ACCEPT

[root @www ~]# iptables -A INPUT -i eth0 -p tcp --dport 139 -j ACCEPT

[root@www ~]# iptables -A INPUT -i eth0 -p tcp --dport 445 -j ACCEPT

例如：只要来自 192.168.1.0/24 的1024:65535 埠口的封包，且想要连线到本机的ssh port 就予以抵挡，可以这样做：

[root@www ~]# iptables -A INPUT -i eth0 -p tcp -s 192.168.1.0/24 \

> --sport 1024:65534 --dport ssh -j DROP

范例：将来自任何地方来源port 1:1023的主动连线到本机端的1:1023连线丢弃

[root@www ~]# iptables -A INPUT -i eth0 -p tcp --sport 1:1023 \

> --dport 1:1023 --syn -j DROP

外挂模块 mac state

[root@www ~]# iptables -A INPUT [-m state] [--state状态]

选项与参数：

-m ：一些iptables 的外挂模组，主要常见的有：

state ：状态模组

mac ：网路卡硬体位址(hardware address)

--state ：一些封包的状态，主要有：

INVALID ：无效的封包，例如资料破损的封包状态

ESTABLISHED：已经连线成功的连线状态；

NEW ：想要新建立连线的封包状态；

RELATED ：这个最常用！表示这个封包是与我们主机发送出去的封包有关

范例：只要已建立或相关封包就予以通过，只要是不合法封包就丢弃

[root@www ~]# iptables -A INPUT -m state \

> --state RELATED,ESTABLISHED -j ACCEPT

[root@www ~] # iptables -A INPUT -m state --state INVALID -j DROP

范例：针对区域网路内的aa:bb:cc:dd:ee:ff主机开放其连线

[root@www ~]# iptables -A INPUT -m mac --mac-source aa:bb:cc:dd :ee:ff \

> -j ACCEPT

选项与参数：

--mac-source ：就是来源主机的MAC 啦！

#可用来针对内部有人改ip来上网,DROP掉它

20200616

磁盘配额(quota)与高级文件系统管理

磁盘配额

一般用途,针对网络服务

针对网站,每个人的网页空间的容量限制

针对邮件服务器,每个人的邮件空间限制

针对文件服务器,每个人最大的可用网络硬盘空间

针对linux系统主机

限制某一用户组所能使用的最大磁盘配额，类似收费会员空间会更大

限制某一用户的最大磁盘配额

限制某一目录的最大磁盘配额

旧的ext文件系统主要是针对整个文件系统来处理,新的xfs可以使用project模式针对个别的目录来配额

使用限制

ext文件仅能针对整个文件系统

内核必须支持磁盘配额

只对一般用户有效,root无效

若启用selinux,并非所有目录可设置磁盘配额

针对xfs文件系统

分别针对用户，用户组或个别目录

容量限制或文件数量限制(block或inode)

软限制与硬限制

如sort 400M, hard 500M,低过400正常使用,超过500系统锁住磁盘使用权，用户在400~500时每登录系统时,系统会主动发出磁盘容量将耗尽的信息,且给予一个宽限时间(默认7天),7天内不进行磁盘管理,soft的值会替换hard值，磁盘使用权会被锁住无法新增文件

实例

目的与帐号：现在我想要让我的专题生五个为一组，这五个人的帐号分别是myquota1, myquota2, myquota3, myquota4, myquota5，这五个用户的密码都是password ，且这五个用户所属的初始群组都是myquotagrp 。其他的帐号属性则使用预设值。

帐号的磁碟容量限制值：我想让这五个用户都能够取得300MBytes 的磁碟使用量(hard)，档案数量则不予限制。此外，只要容量使用率超过250MBytes ，就予以警告(soft)。

群组的限额(option 1)：由于我的系统里面还有其他用户存在，因此我仅承认myquotagrp 这个群组最多仅能使用1GBytes 的容量。这也就是说，如果myquota1, myquota2, myquota3 都用了280MBytes 的容量了，那么其他两人最多只能使用 (1000MB - 280x3 = 160MB) 的磁碟容量啰！这就是使用者与群组同时设定时会产生的后果。

共享目录限额(option 2)：另一种设定方式，每个用户还是具有自己独立的容量限止，但是这五个人的专题共用目录在/home/myquota这里，该目录请设定为其他人没有任何权限的共享目录空间，仅有myquotagrp群组拥有全部的权限。且无论如何，该目录最多仅能够接受500MBytes的容量。请注意，群组(group)的限制与目录(directory/project)无法同时并存喔！所以底下的流程中，我们会先以群组来设计，然后再以目录限制来进一步说明！

宽限时间的限制：最后，我希望每个使用者在超过soft 限制值之后，都还能够有14 天的宽限时间。

#制作帐号环境时，由于有五个帐号，因此鸟哥使用script来建立环境！

[root@study ~]# vim addaccount.sh

#!/bin/bash

# 使用script 来建立实验quota 所需的环境

groupadd myquotagrp

for username in myquota1 myquota2 myquota3 myquota4 myquota5

do

useradd -g myquotagrp $username

echo "password" | passwd --stdin $username

done

mkdir /home/myquota

chgrp myquotagrp /home/myquota

chmod 2770 /home/myquota

[root@study ~]# sh addaccount.sh

档案系统的支援与观察

[root@study ~]# vim /etc/fstab

/dev/mapper/centos-home /home xfs defaults ,usrquota,grpquota 0 0

#其他项目鸟哥并没有列出来！重点在于第四栏位！于default后面加上两个参数！

[root@study ~]# umount /home

[root@study ~]# mount -a

[root@study ~]# mount | grep home

/dev/mapper/centos-home on /home type xfs (rw,relatime,seclabel ,attr2,inode64, usrquota,grpquota )

基本上，针对quota 限制的项目主要有三项，如下所示：

uquota/usrquota/quota：针对使用者帐号的设定

gquota/grpquota：针对群组的设定

pquota/prjquota：针对单一目录的设定，但是不可与grpquota 同时存在！

Quota 流程-2：观察Quota 报告资料

[root@study ~]# xfs\_quota -x -c "指令" [挂载点]

选项与参数：

-x ：专家模式，后续才能够加入-c 的指令参数喔！

-c ：后面加的就是指令，这个小节我们先来谈谈数据回报的指令

指令：

print ：单纯的列出目前主机内的档案系统参数等资料

df ：与原本的df 一样的功能，可以加上-b (block) -i (inode) -h (加上单位) 等

report：列出目前的quota 项目，有-ugr (user/group/project) 及-bi 等资料

state ：说明目前支援quota 的档案系统的资讯，有没有起动相关项目等

范例一：列出目前系统的各的档案系统，以及档案系统的quota挂载参数支援

[root@study ~]# xfs\_quota -x -c "print"

Filesystem Pathname

/ /dev/mapper/centos-root

/srv/myproject /dev/vda4

/boot /dev/vda2

/home /dev/mapper/centos-home ( uquota, gquota ) #所以这里就有显示支援啰

范例二：列出目前/home这个支援quota的载点档案系统使用情况

[root@study ~]# xfs\_quota -x -c "df -h" /home

Filesystem Size Used Avail Use% Pathname

/dev/mapper/centos-home

5.0G 67.0M 4.9G 1% /home

# 如上所示，其实跟原本的df 差不多啦！只是会更正确就是了。

范例三：列出目前/home的所有用户的quota限制值

[root@study ~]# xfs\_quota -x -c "report -ubih" /home

User quota on /home (/dev/mapper/centos-home)

Blocks Inodes

User ID Used Soft Hard Warn/Grace Used Soft Hard Warn/Grace

---------- --------------------------------- ------- --------------------------

root 4K 0 0 00 [------] 4 0 0 00 [------]

dmtsai 34.0M 0 0 00 [------] 432 0 0 00 [------]

.....(中间省略).....

myquota1 12K 0 0 00 [------] 7 0 0 00 [------]

myquota2 12K 0 0 00 [------] 7 0 0 00 [------]

myquota3 12K 0 0 00 [------] 7 0 0 00 [------]

myquota4 12K 0 0 00 [------] 7 0 0 00 [------]

myquota5 12K 0 0 00 [------] 7 0 0 00 [------]

# 所以列出了所有用户的目前的档案使用情况，并且列出设定值。注意，最上面的Block

# 代表这个是block 容量限制，而inode 则是档案数量限制喔。另外，soft/hard 若为0，代表没限制

范例四：列出目前支援的quota档案系统是否有起动了quota功能？

[root@study ~]# xfs\_quota -x -c "state"

User quota state on /home (/dev/mapper/centos-home)

Accounting: ON #有启用计算功能

Enforcement: ON #有实际quota管制的功能

Inode: #1568 (4 blocks, 4 extents) #上面四行说明的是有启动user的限制能力

Group quota state on /home (/dev/mapper/centos-home)

Accounting: ON

Enforcement: ON

Inode: #1569 (5 blocks, 5 extents) #上面四行说明的是有启动group的限制能力

Project quota state on /home (/dev/mapper/centos-home)

Accounting: OFF

Enforcement: OFF

Inode: #1569 (5 blocks, 5 extents) #上面四行说明的是project并未支援

Blocks grace time: [7 days 00:00:30] #底下则是grace time的项目

Inodes grace time: [7 days 00:00:30]

Realtime Blocks grace time: [7 days 00:00:30]

Quota 流程-3：限制值设定方式

[root@study ~]# xfs\_quota -x -c "limit [-ug] b[soft|hard]=N i[soft|hard]=N name"

[root@study ~]# xfs\_quota -x -c "timer [-ug] [-bir] Ndays"

选项与参数：

limit ：实际限制的项目，可以针对user/group 来限制，限制的项目有

bsoft/bhard : block 的soft/hard 限制值，可以加单位

isoft/ihard : inode 的soft/hard 限制值

name : 就是用户/群组的名称啊！

timer ：用来设定grace time 的项目喔，也是可以针对user/group 以及block/inode 设定

范例一：设定好用户们的block限制值(题目中没有要限制inode啦！)

[root@study ~]# xfs\_quota -x -c "limit -u bsoft=250M bhard=300M myquota1" /home

[root@ study ~]# xfs\_quota -x -c "limit -u bsoft=250M bhard=300M myquota2" /home

[root@study ~]# xfs\_quota -x -c "limit -u bsoft=250M bhard=300M myquota3" /home

[ root@study ~]# xfs\_quota -x -c "limit -u bsoft=250M bhard=300M myquota4" /home

[root@study ~]# xfs\_quota -x -c "limit -u bsoft=250M bhard=300M myquota5" / home

[root@study ~]# xfs\_quota -x -c "report -ubih" /home

User quota on /home (/dev/mapper/centos-home)

Blocks Inodes

User ID Used Soft Hard Warn/Grace Used Soft Hard Warn/Grace

---------- --------------------------------- ------- --------------------------

myquota1 12K 250M 300M 00 [------] 7 0 0 00 [------]

范例二：设定好myquotagrp的block限制值

[root@study ~]# xfs\_quota -x -c "limit -g bsoft=950M bhard=1G myquotagrp" /home

[root@study ~]# xfs\_quota -x -c "report -gbih" /home

Group quota on /home (/dev/mapper/centos-home)

Blocks Inodes

Group ID Used Soft Hard Warn/Grace Used Soft Hard Warn/Grace

---------- --------------------------------- ------- --------------------------

myquotagrp 60K 950M 1G 00 [------] 36 0 0 00 [------]

范例三：设定一下grace time变成14天吧！

[root@study ~]# xfs\_quota -x -c "timer -ug -b 14days" /home

[root@study ~]# xfs\_quota -x -c "state" /home

User quota state on /home (/dev/mapper/centos-home)

.....(中间省略).....

Blocks grace time: [ 14 days 00:00:30]

Inodes grace time: [7 days 00:00:30]

Realtime Blocks grace time: [7 days 00:00:30]

范例四：以myquota1用户测试quota是否真的实际运作呢？

[root@study ~]# su - myquota1

[myquota1@study ~]$ dd if=/dev/zero of=123.img bs=1M count=310

dd: error writing '123.img': Disk quota exceeded

300+0 records in

299+0 records out

314552320 bytes (315 MB) copied, 0.181088 s, 1.7 GB/s

[myquota1@study ~]$ ll -h

-rw-r--r--. 1 myquota1 myquotagrp 300M Jul 24 21:38 123.img

[myquota1@study ~]$ exit

[root@study ~]# xfs\_quota -x -c "report -ubh" /home

User quota on /home (/dev/mapper/centos-home)

Blocks

User ID Used Soft Hard Warn/Grace

---------- ---------------------------------

myquota1 300M 250M 300M 00 [13 days]

myquota2 12K 250M 300M 00 [------]

# 因为myquota1 的磁碟用量已经破表，所以当然就会出现那个可怕的grace time 啰！

针对目录的限制参考鸟哥

不修改即有系统的磁盘配额

如要对/var/spool/mail邮件进行限额

将/var/spool/mail 目录完整地移动到/home下

利用ln -s /home/mail /var/spool/mail 建立软链

对/home进行磁盘配额

磁盘阵列(RAID)

独立冗余磁盘阵列,RAID通过技术(软件或硬件)将多个较小的磁盘整合成为一个较大的磁盘设备,磁盘不单可以存储而且还具有数据保护的功能

常见level

RAID 0 (等量模式,stripe)，性能最佳

这模式会将磁盘先切出先切出等量的数据块(chunk,4K~1M),然后当一个文件写入raid时,该文件会根据chunk的大小切好,再依序放到各磁盘中，因此数据写入RAID时,数据会等量地放置在各个磁盘上面(如有100M数写入两块磁盘组成的RAID时,各磁盘被分配50M的数据).磁盘使用相同型号与容量组成时效果较佳.越多块磁盘组成的RAI

D 0性能越好,因为每块负责的数据量就更低,但RAID 0只要有任何一块磁盘损坏,在RAID上的数据都会遗失

RAID 1(镜像模式, mirror),完整备份

也是最好相同型号相同的磁盘,如果是不同容量磁盘组成,那总容量将以最小的那块为主.这模式是让同一份数据完整地保存在两块磁盘上，如100M文件写入两块磁盘组成的RAID 1，那么(通过I/0总线(南桥),因此写的性能差)这两块磁盘将会同步写入100M到它们上面,因此整体的RAID的容量几乎少了50%.如果是硬件raid 1，磁盘阵列卡就不使用I/O总线(性能较好).优点,任何一块磁盘损坏时数据还是完整地保存下来,缺点磁盘的容量有一半在备份

RAID 10 , RAID 01

所谓的RAID10就是先让两块磁盘组成RAID 1，并且这样的设置共有两组,然后将两组RAID1组成一组RAID0,推荐10,如有20块磁盘组成的,每两块组成RAID1,这样就总共有10组可以自已恢复系统,然后10组再组成一个raid0，速度立刻提升了10倍

RAID 5 性能与数据备份的均衡考虑

需要三块以上的磁盘,数据循环写入时会有部分的奇偶校验值(parity)被纪录上来,并且都记录在不同的磁盘,因此任何一个磁盘损坏时都能够借由其它的磁盘的检查码来重建原本磁盘内的数据，由于有奇偶检验值,因此raid5的总容量会是整体磁盘数量减一块。如3块磁盘只会剩下2块的容量,而且当损坏的磁盘数量大于等于两块时,raid5数据就会被损坏,因此仅能支持一块磁盘的损坏

RAID6

与RAID5相似,不过RAID6使用两块磁盘的容量存储奇偶校验值,因此磁盘容量就会少两块,但是允许出错的磁盘数量就可以达到两块了

热备份磁盘(spare disk)

热备份磁盘是一块或多块没有包含在原本磁盘阵列中的磁盘,平时并不会使用,当磁盘阵列有任何磁盘损坏时,这块热备份磁盘就会被主动拉进磁盘中,并将坏掉的那块硬盘移动磁盘阵列,然后立即重建数据系统，数据就会安全继续保存着

磁盘阵列的优点

数据的安全与可靠

读写性能,如RAID0可加强读写性能,让系统的I/O部分得到改善

容量,可以让多块磁盘组合起来提升容量

硬件RAID，软件RAID

硬件RAID就是通过磁盘阵列卡(有芯片)来完成RAID的任务

软件RAID是通过软件技术来模拟RAID任务

软件RAID的设置

[root@study ~]# mdadm --detail /dev/md0

[root@study ~]# mdadm --create /dev/md[0-9] --auto=yes --level=[015] --chunk =NK \

> --raid-devices=N --spare-devices=N /dev/sdx /dev/hdx...

选项与参数：

--create ：为建立RAID 的选项；

--auto=yes ：决定建立后面接的软体磁碟阵列装置，亦即/dev/md0, /dev/md1...

--chunk=Nk ：决定这个装置的chunk 大小，也可以当成stripe 大小，一般是64K 或512K。

--raid-devices=N ：使用几个磁碟(partition) 作为磁碟阵列的装置

--spare-devices=N ：使用几个磁碟作为备用(spare) 装置

--level=[015] ：设定这组磁碟阵列的等级。支援很多，不过建议只要用0, 1, 5 即可

--detail ：后面所接的那个磁碟阵列装置的详细资讯

例

利用4 个partition 组成RAID 5；

每个partition 约为1GB 大小，需确定每个partition 一样大较佳；

利用1 个partition 设定为spare disk

chunk 设定为256K 这么大即可！

这个spare disk 的大小与其他RAID 所需partition 一样大！

将此RAID 5 装置挂载到/srv/raid 目录下

[root@study ~]# lsblk

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

vda 252:0 0 40G 0 disk

|-vda1 252:1 0 2M 0 part

|-vda2 252:2 0 1G 0 part /boot

|-vda3 252:3 0 30G 0 part

| |-centos-root 253:0 0 10G 0 lvm /

| |-centos-swap 253:1 0 1G 0 lvm [SWAP]

| `-centos-home 253:2 0 5G 0 lvm /home

|-vda4 252:4 0 1G 0 part /srv/myproject

|-vda5 252:5 0 1G 0 part

|-vda6 252:6 0 1G 0 part

|-vda7 252:7 0 1G 0 part

|-vda8 252:8 0 1G 0 part

`-vda9 252:9 0 1G 0 part #5块分区,一般是磁盘

以mdadm 建置RAID

接下来就简单啦！透过mdadm 来建立磁碟阵列先！

[root@study ~]# mdadm --create /dev/md0 --auto=yes --level=5 --chunk=256K \

> --raid-devices=4 --spare-devices=1 /dev/vda {5,6,7,8,9}

mdadm: /dev/vda5 appears to contain an ext2fs file system

size=1048576K mtime=Thu Jun 25 00:35:01 2015 #某些时刻会出现这个东西！没关系的！

Continue creating array? y

mdadm: Defaulting to version 1.2 metadata

mdadm: array /dev/md0 started.

# 详细的参数说明请回去前面看看啰！这里我透过{} 将重复的项目简化！

# 此外，因为鸟哥这个系统经常在建置测试的环境，因此系统可能会抓到之前的filesystem

# 所以就会出现如上前两行的讯息！那没关系的！直接按下y 即可删除旧系统

[root@study ~]# mdadm --detail /dev/md0

/dev/md0: # RAID的装置档名

Version : 1.2

Creation Time : Mon Jul 27 15:17:20 2015 #建置RAID的时间

Raid Level : raid5 #这就是RAID5等级！

Array Size : 3142656 (3.00 GiB 3.22 GB) #整组RAID的可用容量

Used Dev Size : 1047552 (1023.17 MiB 1072.69 MB) #每颗磁碟(装置)的容量

Raid Devices : 4 #组成RAID的磁碟数量

Total Devices : 5 #包括spare的总磁碟数

Persistence : Superblock is persistent

Update Time : Mon Jul 27 15:17:31 2015

State : clean #目前这个磁碟阵列的使用状态

Active Devices : 4 #启动(active)的装置数量

Working Devices : 5 #目前使用于此阵列的装置数

Failed Devices : 0 #损坏的装置数

Spare Devices : 1 #预备磁碟的数量

Layout : left-symmetric

Chunk Size : 256K #就是chunk的小区块容量

Name : study.centos.vbird:0 (local to host study.centos.vbird)

UUID : 2256da5f:4870775e:cf2fe320:4dfabbc6

Events : 18

Number Major Minor RaidDevice State

0 252 5 0 active sync /dev/vda5

1 252 6 1 active sync /dev/vda6

2 252 7 2 active sync /dev/vda7

5 252 8 3 active sync /dev/vda8

4 252 9 - spare /dev/vda9

# 最后五行就是这五个装置目前的情况，包括四个active sync 一个spare ！

# 至于RaidDevice 指的则是此RAID 内的磁碟顺序

格式化与挂载使用RAID

接下来就是开始使用格式化工具啦！这部分就需要注意喔！因为涉及到xfs 档案系统的优化！还记得第七章的内容吧？我们这里的参数为：

srtipe (chunk) 容量为256K，所以su=256k

共有4 颗组成RAID5 ，因此容量少一颗，所以sw=3 喔！

由上面两项计算出资料宽度为： 256K\*3=768k

所以整体来说，要优化这个XFS 档案系统就变成这样：

[root@study ~]# mkfs.xfs -f -d su=256k,sw=3 -r extsize=768k /dev/md0

#有趣吧！是/dev/md0做为装置被格式化呢！

[root@study ~]# mkdir /srv/raid

[root@study ~]# mount /dev/md0 /srv/raid

[root@study ~]# df -Th /srv/raid

Filesystem Type Size Used Avail Use% Mounted on

/dev/md0 xfs 3.0G 33M 3.0G 2% /srv/raid

# 看吧！多了一个/dev/md0 的装置，而且真的可以让你使用呢！还不赖！

模拟RAID 错误的救援模式

[root@study ~]# mdadm --manage /dev/md[0-9] [--add装置] [--remove装置] [--fail装置]

选项与参数：

--add ：会将后面的装置加入到这个md 中！

--remove ：会将后面的装置由这个md 中移除

--fail ：会将后面的装置设定成为出错的状态

设定磁碟为错误(fault)

首先，我们来处理一下，该如何让一个磁碟变成错误，然后让spare disk 自动的开始重建系统呢？

# 0.先复制一些东西到/srv/raid去，假设这个RAID已经在使用了

[root@study ~]# cp -a /etc /var/log /srv/raid

[root@study ~]# df - Th /srv/raid ; du -sm /srv/raid/\*

Filesystem Type Size Used Avail Use% Mounted on

/dev/md0 xfs 3.0G 144M 2.9G 5% /srv/raid

28 /srv/raid/etc <==看吧！确实有资料在里面喔！

51 /srv/raid/log

# 1.假设/dev/vda7这个装置出错了！实际模拟的方式：

[root@study ~]# mdadm --manage /dev/md0 --fail /dev/vda7

mdadm: set /dev/vda7 faulty in /dev/md0 #设定成为错误的装置啰！

/dev/md0:

.....(中间省略).....

Update Time : Mon Jul 27 15:32:50 2015

State : clean, degraded, recovering

Active Devices : 3

Working Devices : 4

Failed Devices : 1 <==出错的磁碟有一个！

Spare Devices : 1

.....(中间省略).....

Number Major Minor RaidDevice State

0 252 5 0 active sync /dev/vda5

1 252 6 1 active sync /dev/vda6

4 252 9 2 spare rebuilding /dev/vda9

5 252 8 3 active sync /dev/vda8

2 252 7 - faulty /dev/vda7

#看到没！这的动作要快做才会看到！/dev/vda9启动了而/dev/vda7死掉了

上面的画面你得要快速的连续输入那些mdadm 的指令才看的到！因为你的RAID 5 正在重建系统！若你等待一段时间再输入后面的观察指令，则会看到如下的画面了：

# 2.已经藉由spare disk重建完毕的RAID 5情况

[root@study ~]# mdadm --detail /dev/md0

....(前面省略)....

Number Major Minor RaidDevice State

0 252 5 0 active sync /dev/vda5

1 252 6 1 active sync /dev/vda6

4 252 9 2 active sync /dev/vda9

5 252 8 3 active sync /dev/vda8

2 252 7 - faulty /dev/vda7

看吧！又恢复正常了！真好！我们的/srv/raid 档案系统是完整的！并不需要卸载！很棒吧！

将出错的磁碟移除并加入新磁碟

因为我们的系统那个/dev/vda7 实际上没有坏掉啊！只是用来模拟而已啊！因此，如果有新的磁碟要替换，其实替换的名称会一样啊！也就是我们需要：

先从/dev/md0 阵列中移除/dev/vda7 这颗『磁碟』

整个Linux 系统关机，拔出/dev/vda7 这颗『磁碟』，并安装上新的/dev/vda7 『磁碟』，之后开机

将新的/dev/vda7 放入/dev/md0 阵列当中！

# 3.拔除『旧的』/dev/vda7磁碟

[root@study ~]# mdadm --manage /dev/md0 --remove /dev/vda7

#假设接下来你就进行了上面谈到的第2 , 3个步骤，然后重新开机成功了！

# 4.安装『新的』/dev/vda7磁碟

[root@study ~]# mdadm --manage /dev/md0 --add /dev/vda7

[root@study ~]# mdadm --detail /dev/ md0

....(前面省略)....

Number Major Minor RaidDevice State

0 252 5 0 active sync /dev/vda5

1 252 6 1 active sync /dev/vda6

4 252 9 2 active sync /dev/vda9

5 252 8 3 active sync /dev/vda8

6 252 7 - spare /dev/vda7

开机自动启动RAID 并自动挂载

[root@study ~]# mdadm --detail /dev/md0 | grep -i uuid

UUID : 2256da5f:4870775e:cf2fe320:4dfabbc6

# 后面那一串资料，就是这个装置向系统注册的UUID 识别码！

#开始设定mdadm.conf

[root@study ~]# vim /etc/mdadm.conf

ARRAY /dev/md0 UUID=2256da5f:4870775e:cf2fe320:4dfabbc6

# RAID装置识别码内容

#开始设定开机自动挂载并测试

[root@study ~]# blkid /dev/md0

/dev/md0: UUID="494cb3e1-5659-4efc-873d-d0758baec523" TYPE="xfs"

[root@study ~]# vim /etc/fstab

UUID=494cb3e1-5659-4efc-873d-d0758baec523 /srv/raid xfs defaults 0 0

[root@study ~]# umount /dev/md0; mount -a

[root@study ~]# df -Th /srv/raid

Filesystem Type Size Used Avail Use% Mounted on

/dev/md0 xfs 3.0G 111M 2.9G 4% /srv/raid

# 你得确定可以顺利挂载，并且没有发生任何错误！

关闭raid

# 1.先卸载且删除设定档内与这个/dev/md0有关的设定：

[root@study ~]# umount /srv/raid

[root@study ~]# vim /etc/fstab

UUID=494cb3e1-5659- 4efc-873d-d0758baec523 /srv/raid xfs defaults 0 0

#将这一行删除掉！或者是注解掉也可以！

# 2.先覆盖掉RAID的metadata以及XFS的superblock，才关闭/dev/md0的方法

[root@study ~]# dd if=/dev/zero of=/dev/md0 bs=1M count=50

[root @study ~]# mdadm --stop /dev/md0

mdadm: stopped /dev/md0 <==不啰唆！这样就关闭了！

[root@study ~]# dd if=/dev/zero of=/dev/vda5 bs=1M count=10

[root@study ~]# dd if=/dev/zero of=/dev/vda6 bs=1M count =10

[root@study ~]# dd if=/dev/zero of=/dev/vda7 bs=1M count=10

[root@study ~]# dd if=/dev/zero of=/dev/vda8 bs= 1M count=10

[root@study ~]# dd if=/dev/zero of=/dev/vda9 bs=1M count=10

[root@study ~]# cat /proc/mdstat

Personalities : [raid6] [raid5] [raid4]

unused devices: <none> <==看吧！确实不存在任何阵列装置！

[root@study ~]# vim /etc/mdadm.conf

#ARRAY /dev/md0 UUID=2256da5f:4870775e:cf2fe320:4dfabbc6

#一样啦！删除他或是注解他！

LVM(逻缉卷管理器)

弹性调整系统的容量,并不是性能与数据安全的考量

LVM可以整合多个物理分区,让这些分区看起来像一个磁盘一样(VG),然后将这块大磁盘经过划分为可使用的分区(lv)最后挂载使用,未来还可以在LVM中新增或删除其他的物理分区

物理卷(PV)

实际的分区或磁盘需要调整系统标识符成为LVM标识符(不修改也没关系)(8e),然后通过pvcreate命令将它转成LVM最底层的镜像物理卷(pv)

卷组(VG)

将多个PV整合成VG(看起来像大磁盘)

物理扩展块(PE)

默认4M，LVM最小存储单位,数据都是借由写出PE来完成(类似文件统成的block)

逻辑卷(LV)

最终VG还会切成LV，LV就是最后可以被格式化类似分区的东西,设备各通常为/dev/vgname/vname

LVM是通过交换PE来进行数据转换(修改容量),将原本LV内的PE转移到其它设备中降低容量,或将其它设备的PE加入此LV来增加容量

数据写入LV时有两种方式

线性模式

如/dev/vda1,/dev/vda2两个分区加入到VG中,并且整个VG只有一个LV时,会先用用/dev/vda1的磁盘再用vda2的

交错模式

就是数据分别写入vd1与vd2(类似raid 0 ),性能较好

LVM的主要功能就是弹性调整容量,所以默认是线性模式,不建议使用交错模式

(插入 ssl免费证书 FreeSSL.cn)

LVM 实作流程

使用4 个partition ，每个partition 的容量均为1GB 左右，且system ID 需要为8e；

全部的partition 整合成为一个VG，VG 名称设定为vbirdvg；且PE 的大小为16MB；

建立一个名为vbirdlv 的LV，容量大约2G 好了！

最终这个LV 格式化为xfs 的档案系统，且挂载在/srv/lvm 中

. Disk 阶段(实际的磁碟)

鸟哥就不仔细的介绍实体分割了，请您自行参考第七章的gdisk 来达成底下的范例：

[root@study ~]# gdisk -l /dev/vda

Number Start (sector) End (sector) Size Code Name

1 2048 6143 2.0 MiB EF02

2 6144 2103295 1024.0 MiB 0700

3 2103296 65026047 30.0 GiB 8E00

4 65026048 67123199 1024.0 MiB 8300 Linux filesystem

5 67123200 69220351 1024.0 MiB 8E00 Linux LVM

6 69220352 71317503 1024.0 MiB 8E00 Linux LVM

7 71317504 73414655 1024.0 MiB 8E00 Linux LVM

8 73414656 75511807 1024.0 MiB 8E00 Linux LVM

9 75511808 77608959 1024.0 MiB 8E00 Linux LVM

#其实system ID不改变也没关系！只是为了让我们管理员清楚知道该partition的内容，

# 所以这里建议还是修订成正确的磁碟内容较佳！

. PV 阶段

要建立PV 其实很简单，只要直接使用pvcreate 即可！我们来谈一谈与PV 有关的指令吧！

pvcreate ：将实体partition 建立成为PV ；

pvscan ：搜寻目前系统里面任何具有PV 的磁碟；

pvdisplay ：显示出目前系统上面的PV 状态；

pvremove ：将PV 属性移除，让该partition 不具有PV 属性。

那就直接来瞧一瞧吧！

# 1.检查有无PV在系统上，然后将/dev/vda{5-8}建立成为PV格式

[root@study ~]# pvscan

PV /dev/vda3 VG centos lvm2 [30.00 GiB / 14.00 GiB free]

Total: 1 [30.00 GiB] / in use: 1 [30.00 GiB] / in no VG: 0 [0 ]

# 其实安装的时候，我们就有使用LVM 了喔！所以会有/dev/vda3 存在的！

[root@study ~]# pvcreate /dev/vda{5,6,7,8}

Physical volume "/dev/vda5" successfully created

Physical volume "/dev/vda6" successfully created

Physical volume "/dev/vda7" successfully created

Physical volume "/dev/vda8" successfully created

# 这个指令可以一口气建立这四个partition 成为PV 啦！注意大括号的用途

[root@study ~]# pvscan

PV /dev/vda3 VG centos lvm2 [30.00 GiB / 14.00 GiB free]

PV /dev/vda8 lvm2 [1.00 GiB]

PV /dev/vda5 lvm2 [1.00 GiB]

PV /dev/vda7 lvm2 [1.00 GiB]

PV /dev/vda6 lvm2 [1.00 GiB]

Total: 5 [34.00 GiB] / in use: 1 [30.00 GiB] / in no VG: 4 [4.00 GiB]

# 这就分别显示每个PV 的资讯与系统所有PV 的资讯。尤其最后一行，显示的是：

# 整体PV 的量/ 已经被使用到VG 的PV 量/ 剩余的PV 量

# 2.更详细的列示出系统上面每个PV的个别资讯：

[root@study ~]# pvdisplay /dev/vda5

"/dev/vda5" is a new physical volume of "1.00 GiB"

--- NEW Physical volume ---

PV Name /dev/vda5 <==实际的partition装置名称

VG Name <==因为尚未分配出去，所以空白！

PV Size 1.00 GiB <==就是容量说明

Allocatable NO <==是否已被分配，结果是NO

PE Size 0 <==在此PV内的PE大小

Total PE 0 <==共分割出几个PE

Free PE 0 <==没被LV用掉的PE

Allocated PE 0 <==尚可分配出去的PE数量

PV UUID Cb717z-lShq-6WXf-ewEj-qg0W-MieW-oAZTR6

# 由于PE 是在建立VG 时才给予的参数，因此在这里看到的PV 里头的PE 都会是 0

# 而且也没有多余的PE 可供分配(allocatable)。

2. VG 阶段

建立VG 及VG 相关的指令也不少，我们来看看：

vgcreate ：就是主要建立VG 的指令啦！他的参数比较多，等一下介绍。

vgscan ：搜寻系统上面是否有VG 存在？

vgdisplay ：显示目前系统上面的VG 状态；

vgextend ：在VG 内增加额外的PV ；

vgreduce ：在VG 内移除PV；

vgchange ：设定VG 是否启动(active)；

vgremove ：删除一个VG 啊！

与PV 不同的是， VG 的名称是自订的！我们知道PV 的名称其实就是partition 的装置档名， 但是这个VG 名称则可以随便你自己取啊！在底下的例子当中，我将VG 名称取名为vbirdvg 。建立这个 VG 的流程是这样的：

[root@study ~]# vgcreate [-s N[mgt]] VG名称PV名称

选项与参数：

-s ：后面接PE 的大小(size) ，单位可以是m, g, t (大小写均可)

# 1.将/dev/vda5-7建立成为一个VG，且指定PE为16MB喔！

[root@study ~]# vgcreate -s 16M vbirdvg /dev/vda{5,6,7}

Volume group "vbirdvg" successfully created

[root@study ~]# vgscan

Reading all physical volumes. This may take a while...

Found volume group "vbirdvg" using metadata type lvm2 #我们手动制作的

Found volume group "centos" using metadata type lvm2 #之前系统安装时作的

[root@study ~]# pvscan

PV /dev/vda5 VG vbirdvg lvm2 [1008.00 MiB / 1008.00 MiB free]

PV /dev/vda6 VG vbirdvg lvm2 [1008.00 MiB / 1008.00 MiB free]

PV /dev/vda7 VG vbirdvg lvm2 [1008.00 MiB / 1008.00 MiB free]

PV /dev/vda3 VG centos lvm2 [30.00 GiB / 14.00 GiB free]

PV /dev/vda8 lvm2 [1.00 GiB]

Total: 5 [33.95 GiB] / in use: 4 [32.95 GiB] / in no VG: 1 [1.00 GiB]

# 嘿嘿！发现没！有三个PV 被用去，剩下1 个/dev/vda8 的PV 没被用掉！

[root@study ~]# vgdisplay vbirdvg

--- Volume group ---

VG Name vbirdvg

System ID

Format lvm2

Metadata Areas 3

Metadata Sequence No 1

VG Access read/write

VG Status resizable

MAX LV 0

Cur LV 0

Open LV 0

Max PV 0

Cur PV 3

Act PV 3

VG Size 2.95 GiB <==整体的VG容量有这么大

PE Size 16.00 MiB <==内部每个PE的大小

Total PE 189 <==总共的PE数量共有这么多！

Alloc PE / Size 0 / 0

Free PE / Size 189 / 2.95 GiB <==尚可配置给LV的PE数量/总容量有这么多！

VG UUID Rx7zdR-y2cY-HuIZ-Yd2s-odU8-AkTW-okk4Ea

# 最后那三行指的就是PE 能够使用的情况！由于尚未切出LV，因此所有的PE 均可自由使用。

这样就建立一个VG 了！假设我们要增加这个VG 的容量，因为我们还有/dev/vda8 嘛！此时你可以这样做：

# 2.将剩余的PV (/dev/vda8)丢给vbirdvg吧！

[root@study ~]# vgextend vbirdvg /dev/vda8

Volume group "vbirdvg" successfully extended

[root@study ~]# vgdisplay vbirdvg

....(前面省略)....

VG Size 3.94 GiB

PE Size 16.00 MiB

Total PE 252

Alloc PE / Size 0 / 0

Free PE / Size 252 / 3.94 GiB

# 基本上，不难吧！这样就可以抽换整个VG 的大小啊！

3. LV 阶段

创造出VG 这个大磁碟之后，再来就是要建立分割区啦！这个分割区就是所谓的LV 啰！假设我要将刚刚那个 vbirdvg 磁碟，分割成为vbirdlv ，整个VG 的容量都被分配到vbirdlv 里面去！先来看看能使用的指令后，就直接工作了先！

lvcreate ：建立LV 啦！

lvscan ：查询系统上面的LV ；

lvdisplay ：显示系统上面的LV 状态啊！

lvextend ：在LV 里面增加容量！

lvreduce ：在LV 里面减少容量；

lvremove ：删除一个LV ！

lvresize ：对LV 进行容量大小的调整！

[root@study ~]# lvcreate [-LN[mgt]] [-n LV名称] VG名称

[root@study ~]# lvcreate [-l N] [-n LV名称] VG名称

选项与参数：

-L ：后面接容量，容量的单位可以是M,G,T 等，要注意的是，最小单位为PE，

因此这个数量必须要是PE 的倍数，若不相符，系统会自行计算最相近的容量。

-l ：后面可以接PE 的『个数』，而不是数量。若要这么做，得要自行计算PE 数。

-n ：后面接的就是LV 的名称啦！

更多的说明应该可以自行查阅吧！man lvcreate

# 1.将vbirdvg分2GB给vbirdlv喔！

[root@study ~]# lvcreate -L 2G -n vbirdlv vbirdvg

Logical volume "vbirdlv" created

# 由于本案例中每个PE 为16M ，如果要用PE 的数量来处理的话，那使用下面的指令也OK喔！

# lvcreate -l 128 -n vbirdlv vbirdvg

[root@study ~]# lvscan

ACTIVE '/dev/vbirdvg/vbirdlv' [2.00 GiB] inherit <==新增加的一个LV啰！

ACTIVE '/dev/centos/root' [10.00 GiB] inherit

ACTIVE '/dev/centos/home' [5.00 GiB] inherit

ACTIVE '/dev/centos/swap' [1.00 GiB] inherit

[root@study ~]# lvdisplay /dev/vbirdvg/vbirdlv

--- Logical volume ---

LV Path /dev/vbirdvg/vbirdlv #这个是LV的全名喔！

LV Name vbirdlv

VG Name vbirdvg

LV UUID QJJrTC-66sm-878Y-o2DC-nN37-2nFR-0BwMmn

LV Write Access read/write

LV Creation host, time study.centos.vbird, 2015-07-28 02:22:49 +0800

LV Status available

# open 0

LV Size 2.00 GiB #容量就是这么大！

Current LE 128

Segments 3

Allocation inherit

Read ahead sectors auto

- currently set to 8192

Block device 253:3

档案系统阶段

这个部分鸟哥我就不再多加解释了！直接来进行吧！

# 1.格式化、挂载与观察我们的LV吧！

[root@study ~]# mkfs.xfs /dev/vbirdvg/vbirdlv <==注意LV全名！

[root@study ~]# mkdir /srv/lvm

[root@study ~]# mount /dev/vbirdvg/vbirdlv /srv/lvm

[root@study ~]# df -Th /srv/lvm

Filesystem Type Size Used Avail Use% Mounted on

/dev/mapper/vbirdvg-vbirdlv xfs 2.0G 33M 2.0G 2% /srv/lvm

[root@study ~]# cp -a /etc /var/log /srv/lvm

[root@study ~]# df -Th /srv/lvm

Filesystem Type Size Used Avail Use% Mounted on

/dev/mapper/vbirdvg-vbirdlv xfs 2.0G 152M 1.9G 8% /srv/lvm <==确定是可用的啊！

透过这样的功能，我们现在已经建置好一个LV 了！你可以自由的应用/srv/lvm 内的所有资源！

14.3.3 放大LV 容量

我们不是说LVM最大的特色就是弹性调整磁碟容量吗？好！那我们就来处理一下，如果要放大LV的容量时，该如何进行完整的步骤呢？其实一点都不难喔！如果你回去看图14.3.2的话，那么你会知道放大档案系统时，需要底下这些流程的：

VG阶段需要有剩余的容量：因为需要放大档案系统，所以需要放大LV，但是若没有多的VG容量，那么更上层的LV与档案系统就无法放大的。因此，你得要用尽各种方法来产生多的VG容量才行。一般来说，如果VG容量不足，最简单的方法就是再加硬碟！然后将该硬碟使用上面讲过的pvcreate及vgextend增加到该VG内即可！

LV阶段产生更多的可用容量：如果VG的剩余容量足够了，此时就可以利用lvresize这个指令来将剩余容量加入到所需要增加的LV装置内！过程相当简单！

档案系统阶段的放大：我们的Linux实际使用的其实不是LV啊！而是LV这个装置内的档案系统！所以一切最终还是要以档案系统为依归！目前在Linux环境下，鸟哥测试过可以放大的档案系统有XFS以及EXT家族！至于缩小仅有EXT家族，目前XFS档案系统并不支援档案系统的容量缩小喔！要注意！要注意！XFS放大档案系统透过简单的xfs\_growfs指令即可！

其中最后一个步骤最重要！我们在第七章当中知道，整个档案系统在最初格式化的时候就建立了inode/block/superblock等资讯，要改变这些资讯是很难的！不过因为档案系统格式化的时候建置的是多个block group ，因此我们可以透过在档案系统当中增加block group的方式来增减档案系统的量！而增减block group就是利用xfs\_growfs啰！所以最后一步是针对档案系统来处理的，前面几步则是针对LVM的实际容量大小！

让我们来实作个范例，假设我们想要针对/srv/lvm 再增加500M 的容量，该如何处置？

# 1.由前面的过程我们知道/srv/lvm是/dev/vbirdvg/vbirdlv这个装置，所以检查vbirdvg吧！

[root@study ~]# vgdisplay vbirdvg

--- Volume group ---

VG Name vbirdvg

System ID

Format lvm2

Metadata Areas 4

Metadata Sequence No 3

VG Access read/write

VG Status resizable

MAX LV 0

Cur LV 1

Open LV 1

Max PV 0

Cur PV 4

Act PV 4

VG Size 3.94 GiB

PE Size 16.00 MiB

Total PE 252

Alloc PE / Size 128 / 2.00 GiB

Free PE / Size 124 / 1.94 GiB #看起来剩余容量确实超过500M的！

VG UUID Rx7zdR-y2cY-HuIZ-Yd2s-odU8-AkTW-okk4Ea

# 既然VG 的容量够大了！所以直接来放大LV 吧！！

# 2.放大LV吧！利用lvresize的功能来增加！

[root@study ~]# lvresize -L +500M /dev/vbirdvg/vbirdlv

Rounding size to boundary between physical extents: 512.00 MiB

Size of logical volume vbirdvg/vbirdlv changed from 2.00 GiB (128 extents) to 2.50 GiB

(160 extents).

Logical volume vbirdlv successfully resized

# 这样就增加了LV 了喔！lvresize 的语法很简单，基本上同样透过-l 或-L 来增加！

# 若要增加则使用+ ，若要减少则使用- ！详细的选项请参考man lvresize 啰！

[root@study ~]# lvscan

ACTIVE '/dev/vbirdvg/vbirdlv' [2.50 GiB] inherit

ACTIVE '/dev/centos/root' [10.00 GiB] inherit

ACTIVE '/dev/centos/home' [5.00 GiB] inherit

ACTIVE '/dev/centos/swap' [1.00 GiB] inherit

# 可以发现/dev/vbirdvg/vbirdlv 容量由2G 增加到2.5G 啰！

[root@study ~]# df -Th /srv/lvm

Filesystem Type Size Used Avail Use% Mounted on

/dev/mapper/vbirdvg-vbirdlv xfs 2.0G 111M 1.9G 6% /srv/lvm

看到了吧？最终的结果中LV 真的有放大到2.5GB 喔！但是档案系统却没有相对增加！而且，我们的LVM 可以线上直接处理，并不需要特别给他umount 哩！真是人性化！但是还是得要处理一下档案系统的容量啦！开始观察一下档案系统，然后使用xfs\_growfs 来处理一下吧！

# 3.1先看一下原本的档案系统内的superblock记录情况吧！

[root@study ~]# xfs\_info /srv/lvm

meta-data=/dev/mapper/vbirdvg-vbirdlv isize=256 agcount=4 , agsize=131072 blks

= sectsz=512 attr=2, projid32bit=1

= crc=0 finobt=0

data = bsize=4096 blocks=524288 , imaxpct=25

= sunit=0 swidth=0 blks

naming =version 2 bsize=4096 ascii-ci=0 ftype=0

log =internal bsize=4096 blocks=2560, version=2

= sectsz=512 sunit=0 blks, lazy-count=1

realtime =none extsz=4096 blocks=0, rtextents=0

[root@study ~]# xfs\_growfs /srv/lvm #这一步骤才是最重要的！

[root@study ~]# xfs\_info /srv/lvm

meta-data=/dev/mapper/vbirdvg-vbirdlv isize=256 agcount=5 , agsize=131072 blks

= sectsz=512 attr=2, projid32bit=1

= crc=0 finobt=0

data = bsize=4096 blocks=655360 , imaxpct=25

= sunit=0 swidth=0 blks

naming =version 2 bsize=4096 ascii-ci=0 ftype=0

log =internal bsize=4096 blocks=2560, version=2

= sectsz=512 sunit=0 blks, lazy-count=1

realtime =none extsz=4096 blocks=0, rtextents=0

[root@study ~]# df -Th /srv/lvm

Filesystem Type Size Used Avail Use% Mounted on

/dev/mapper/vbirdvg-vbirdlv xfs 2.5G 111M 2.4G 5% /srv/lvm

[root@study ~]# ls -l /srv/lvm

drwxr-xr-x. 131 root root 8192 Jul 28 00:12 etc

drwxr-xr-x. 16 root root 4096 Jul 28 00:01 log

# 刚刚复制进去的资料可还是存在的喔！并没有消失不见！

14.3.4 使用LVM thin Volume 让LVM 动态自动调整磁碟使用率

详情查看鸟哥

14.3.5 LVM 的LV 磁碟快照

快照区与被快照的LV必须要在同一个VG上头。

传统快照区的建立

# 1.先观察VG还剩下多少剩余容量

[root@study ~]# vgdisplay vbirdvg

....(其他省略)....

Total PE 252

Alloc PE / Size 226 / 3.53 GiB

Free PE / Size 26 / 416.00 MiB

#就只有剩下26个PE了！全部分配给vbirdsnap1啰！

# 2.利用lvcreate建立vbirdlv的快照区，快照被取名为vbirdsnap1，且给予26个PE

[root@study ~]# lvcreate -s -l 26 -n vbirdsnap1 /dev/vbirdvg/vbirdlv

Logical volume "vbirdsnap1" created

# 上述的指令中最重要的是那个-s 的选项！代表是snapshot 快照功能之意！

# -n 后面接快照区的装置名称， /dev/.... 则是要被快照的LV 完整档名。

# -l 后面则是接使用多少个PE 来作为这个快照区使用。

[root@study ~]# lvdisplay /dev/vbirdvg/vbirdsnap1

--- Logical volume ---

LV Path /dev/vbirdvg/vbirdsnap1

LV Name vbirdsnap1

VG Name vbirdvg

LV UUID I3m3Oc-RIvC-unag-DiiA-iQgI-I3z9-0OaOzR

LV Write Access read/write

LV Creation host, time study.centos.vbird, 2015-07-28 19:21:44 +0800

LV snapshot status active destination for vbirdlv

LV Status available

# open 0

LV Size 2.50 GiB #原始碟，就是vbirdlv的原始容量

Current LE 160

COW-table size 416.00 MiB #这个快照能够纪录的最大容量！

COW-table LE 26

Allocated to snapshot 0.00% #目前已经被用掉的容量！

Snapshot chunk size 4.00 KiB

Segments 1

Allocation inherit

Read ahead sectors auto

- currently set to 8192

Block device 253:11

您看看！这个/dev/vbirdvg/vbirdsnap1 快照区就被建立起来了！而且他的VG 量竟然与原本的/dev/vbirdvg/vbirdlv 相同！也就是说，如果你真的挂载这个装置时，看到的资料会跟原本的vbirdlv 相同喔！我们就来测试看看：

[root@study ~]# mkdir /srv/snapshot1

[root@study ~]# mount -o nouuid /dev/vbirdvg/vbirdsnap1 /srv/snapshot1

[root@study ~]# df -Th /srv/lvm /srv/ snapshot1

Filesystem Type Size Used Avail Use% Mounted on

/dev/mapper/vbirdvg-vbirdlv xfs 2.5G 111M 2.4G 5% /srv/lvm

/dev/mapper/vbirdvg-vbirdsnap1 xfs 2.5G 111M 2.4G 5% /srv/snapshot1

# 有没有看到！这两个咚咚竟然是一模一样喔！我们根本没有动过

# /dev/vbirdvg/vbirdsnap1 对吧！不过这里面会主动记录原vbirdlv 的内容！

因为XFS 不允许相同的UUID 档案系统的挂载，因此我们得要加上那个nouuid 的参数，让档案系统忽略相同的UUID 所造成的问题！没办法啊！因为快照出来的档案系统当然是会一模一样的！

利用快照区复原系统

首先，我们来玩一下，如何利用快照区复原系统吧！不过你要注意的是，你要复原的资料量不能够高于快照区所能负载的实际容量。由于原始资料会被搬移到快照区，如果你的快照区不够大，若原始资料被更动的实际资料量比快照区大，那么快照区当然容纳不了，这时候快照功能会失效喔！

我们的/srv/lvm 已经有/srv/lvm/etc, /srv/lvm/log 等目录了，接下来我们将这个档案系统的内容作个变更， 然后再以快照区资料还原看看：

# 1.先将原本的/dev/vbirdvg/vbirdlv内容作些变更，增增减减一些目录吧！

[root@study ~]# df -Th /srv/lvm /srv/snapshot1

Filesystem Type Size Used Avail Use% Mounted on

/dev/mapper/vbirdvg-vbirdlv xfs 2.5G 111M 2.4G 5% /srv/lvm

/dev/mapper/vbirdvg-vbirdsnap1 xfs 2.5G 111M 2.4G 5% /srv/snapshot1

[root@study ~]# cp -a /usr/share/doc /srv/lvm

[root@study ~]# rm -rf /srv/lvm/log

[root@study ~]# rm -rf /srv/lvm /etc/sysconfig

[root@study ~]# df -Th /srv/lvm /srv/snapshot1

Filesystem Type Size Used Avail Use% Mounted on

/dev/mapper/vbirdvg-vbirdlv xfs 2.5G 146M 2.4G 6% /srv/lvm

/dev/mapper/vbirdvg-vbirdsnap1 xfs 2.5G 111M 2.4G 5% /srv/snapshot1

[root@study ~]# ll /srv/lvm /srv/snapshot1

/srv/lvm:

total 60

drwxr-xr-x. 887 root root 28672 Jul 20 23:03 doc

drwxr-xr-x. 131 root root 8192 Jul 28 00:12 etc

/srv/snapshot1:

total 16

drwxr-xr-x. 131 root root 8192 Jul 28 00:12 etc

drwxr-xr-x. 16 root root 4096 Jul 28 00:01 log

# 两个目录的内容看起来已经不太一样了喔！检测一下快照LV 吧！

[root@study ~]# lvdisplay /dev/vbirdvg/vbirdsnap1

--- Logical volume ---

LV Path /dev/vbirdvg/vbirdsnap1

....(中间省略)....

Allocated to snapshot 21.47%

#鸟哥仅列出最重要的部份！就是全部的容量已经被用掉了21.4%啰！

# 2.利用快照区将原本的filesystem备份，我们使用xfsdump来处理！

[root@study ~]# xfsdump -l 0 -L lvm1 -M lvm1 -f /home/lvm.dump /srv/snapshot1

#此时你就会有一个备份资料，亦即是/home/lvm.dump了！

为什么要备份呢？为什么不可以直接格式化/dev/vbirdvg/vbirdlv 然后将/dev/vbirdvg/vbirdsnap1 直接复制给vbirdlv 呢？要知道vbirdsnap1 其实是vbirdlv 的快照，因此如果你格式化整个vbirdlv 时，原本的档案系统所有资料都会被搬移到vbirdsnap1。那如果vbirdsnap1 的容量不够大(通常也真的不够大)，那么部分资料将无法复制到vbirdsnap1 内，资料当然无法全部还原啊！所以才要在上面表格中制作出一个备份档案的

# 3.将vbirdsnap1卸载并移除(因为里面的内容已经备份起来了)

[root@study ~]# umount /srv/snapshot1

[root@study ~]# lvremove /dev/vbirdvg/vbirdsnap1

Do you really want to remove active logical volume "vbirdsnap1"? [y/n]: y

Logical volume "vbirdsnap1" successfully removed

[root@study ~]# umount /srv/lvm

[root@study ~]# mkfs.xfs -f /dev/vbirdvg/vbirdlv

[root@study ~]# mount /dev/vbirdvg/vbirdlv /srv/lvm

[root @study ~]# xfsrestore -f /home/lvm.dump -L lvm1 /srv/lvm

[root@study ~]# ll /srv/lvm

drwxr-xr-x. 131 root root 8192 Jul 28 00:12 etc

drwxr-xr-x. 16 root root 4096 Jul 28 00:01 log

# 是否与最初的内容相同啊！这就是透过快照来还原的一个简单的方法啰！

LVM 的关闭

先卸载系统上面的LVM 档案系统(包括快照与所有LV)；

使用lvremove 移除LV ；

使用vgchange -an VGname 让VGname 这个VG 不具有Active 的标志；

使用vgremove 移除VG：

使用pvremove 移除PV；

最后，使用fdisk 修改ID 回来啊！

好吧！那就实际的将我们之前建立的所有LVM 资料给删除吧！

[root@study ~]# umount /srv/lvm /srv/thin /srv/snapshot1

[root@study ~]# lvs vbirdvg

LV VG Attr LSize Pool Origin Data% Meta% Move Log Cpy%Sync

vbirdlv vbirdvg -wi-a----- 2.50g

vbirdthin1 vbirdvg Vwi-a-tz-- 10.00g vbirdtpool 4.99

vbirdtpool vbirdvg twi-aotz-- 1.00g 49.93 1.81

# 要注意！先删除vbirdthin1 --> vbirdtpool --> vbirdlv 比较好！

[root@study ~]# lvremove /dev/vbirdvg/vbirdthin1 /dev/vbirdvg/vbirdtpool

[root@study ~]# lvremove /dev/vbirdvg/vbirdlv

[root@study ~]# vgchange -an vbirdvg

0 logical volume(s) in volume group "vbirdvg" now active

[root@study ~]# vgremove vbirdvg

Volume group "vbirdvg" successfully removed

[root@study ~]# pvremove /dev/vda{5,6,7,8}

计划任务(crontab)

centos linux 系统上常见的例行性工作

执行日志文件的轮循

日志文件切割,不然单一文件太大读写性能差,也混乱

日志文件分析

通过logwatch来主动分析各种信息

建立locate的数据库

文件名数据库的更新

manpage查刘数据库的建立

RPM软件日志文件的建立

删除缓存

与网络服务有关的分析操作

如apache的日志

仅执行一次的计划任务

15.2.1 atd 的启动与at 运作的方式

root@study ~]# systemctl restart atd #重新启动atd这个服务

[root@study ~]# systemctl enable atd #让这个服务开机就自动启动

[root@study ~]# systemctl status atd #查阅一下atd目前的状态

atd.service - Job spooling tools

Loaded: loaded (/usr/lib/systemd/system/atd.service; enabled ) #是否开机启动

Active: active (running) since Thu 2015-07-30 19:21:21 CST; 23s ago #是否正在运作中

Main PID: 26503 (atd)

我们可以利用/etc/at.allow 与/etc/at.deny 这两个档案来进行at 的使用限制呢！加上这两个档案后， at 的工作情况其实是这样的：

先找寻/etc/at.allow这个档案，写在这个档案中的使用者才能使用at ，没有在这个档案中的使用者则不能使用at (即使没有写在at.deny当中)；

如果/etc/at.allow不存在，就寻找/etc/at.deny这个档案，若写在这个at.deny的使用者则不能使用at ，而没有在这个at.deny档案中的使用者，就可以使用at咯；

如果两个档案都不存在，那么只有root 可以使用at 这个指令。

[root@study ~]# at [-mldv] TIME

[root@study ~]# at -c工作号码

选项与参数：

-m ：当at 的工作完成后，即使没有输出讯息，亦以email 通知使用者该工作已完成。

-l ：at -l 相当于atq，列出目前系统上面的所有该使用者的at 排程；

-d ：at -d 相当于atrm ，可以取消一个在at 排程中的工作；

-v ：可以使用较明显的时间格式列出at 排程中的工作列表；

-c ：可以列出后面接的该项工作的实际指令内容。

TIME：时间格式，这里可以定义出『什么时候要进行at 这项工作』的时间，格式有：

HH:MM ex> 04:00

在今日的HH:MM 时刻进行，若该时刻已超过，则明天的HH:MM 进行此工作。

HH:MM YYYY-MM-DD ex> 04:00 2015-07-30

强制规定在某年某月的某一天的特殊时刻进行该工作！

HH:MM[am|pm] [Month] [Date] ex> 04pm July 30

也是一样，强制在某年某月某日的某时刻进行！

HH:MM[am|pm] + number [minutes|hours|days|weeks]

ex> now + 5 minutes ex> 04pm + 3 days

就是说，在某个时间点『再加几个时间后』才进行。

范例一：再过五分钟后，将/root/.bashrc寄给root自己

[root@study ~]# at now + 5 minutes <==记得单位要加s喔！

at> /bin/mail -s "testing at job" root < /root/.bashrc

at> <EOT> <==这里输入[ctrl] + d就会出现<EOF>的字样！代表结束！

job 2 at Thu Jul 30 19:35:00 2015

# 上面这行资讯在说明，第2 个at 工作将在2015/07/30 的19:35 进行！

# 而执行at 会进入所谓的at shell 环境，让你下达多重指令等待运作！

范例二：将上述的第2项工作内容列出来查阅

[root@study ~]# at -c 2

#!/bin/sh <==就是透过bash shell的啦！

# atrun uid=0 gid=0

# mail root 0

umask 22

....(中间省略许多的环境变数项目)....

cd /etc/cron\.d || {

echo 'Execution directory inaccessible' >&2

exit 1

}

${SHELL:-/bin/sh} << 'marcinDELIMITER410efc26'

/bin/mail -s "testing at job" root < /root/.bashrc #这一行最重要！

marcinDELIMITER410efc26

#你可以看到指令执行的目录(/root)，还有多个环境变数与实际的指令内容啦！

范例三：由于机房预计于2015/08/05停电，我想要在2015/08/04 23:00关机？

[root@study ~]# at 23:00 2015-08-04

at> /bin/sync

at> /bin/sync

at> /sbin/shutdown -h now

at> <EOT>

job 3 at Tue Aug 4 23:00:00 2015

# 您瞧瞧！at 还可以在一个工作内输入多个指令呢！不错吧！

注意 用绝对路径来下达指令，会是比较一劳永逸的方法(命令与文件)

at的执行与终端机环境无关，而所有standard output/standard error output都会传送到执行者的mailbox去,要输出到终端可用 echo "hello">/dev/tty1

[root@study ~]# atq

[root@study ~]# atrm (jobnumber)

范例一：查询目前主机上面有多少的at工作排程？

[root@study ~]# atq

3 Tue Aug 4 23:00:00 2015 a root

# 上面说的是：『在2015/08/04 的23:00 有一项工作，该项工作指令下达者为

# root』而且，该项工作的工作号码(jobnumber) 为3 号喔！

范例二：将上述的第3个工作移除！

[root@study ~]# atrm 3

[root@study ~]# atq

#没有任何资讯，表示该工作被移除了

batch：系统有空时才进行背景任务,cpu任务负载小于0.8，可用uptime查看

[root@study ~]# batch

at> /usr/bin/updatedb

at> <EOT>

job 4 at Thu Jul 30 19:57:00 2015

循环执行的计划任务

循环执行的许划任务由crond这个系统服务来控制

也有相应的/etc/cron.allow与deny

当用户使用crontab命令来建立计划任务之务,该项任务就会被记录到/var/spool/cron中,且以账号来作为判断

cron执行的每一项任务都会被记录到/var/log/cron日志中,木马会利用crontab来收集志,可以查看些日志看机子是否有木马

[root@study ~]# crontab [-u username] [-l|-e|-r]

选项与参数：

-u ：只有root 才能进行这个任务，亦即帮其他使用者建立/移除crontab 工作排程；

-e ：编辑crontab 的工作内容

-l ：查阅crontab 的工作内容

-r ：移除所有的crontab 的工作内容，若仅要移除一项，请用-e 去编辑。

范例一：用dmtsai的身份在每天的12:00发信给自己

[dmtsai@study ~]$ crontab -e

#此时会进入vi的编辑画面让您编辑工作！注意到，每项工作都是一行。

0 12 \* \* \* mail -s "at 12:00" dmtsai < /home/dmtsai/.bashrc

#分时日月周|<==============指令串=== =====================>|

六个栏位的意义为：

代表意义 分钟 小时 日期 月份 周 指令

数字范围 0-59 0-23 1-31 1-12 0-7 呀就指令啊

比较有趣的是那个『周』喔！周的数字为0 或7 时，都代表『星期天』的意思！另外，还有一些辅助的字符，大概有底下这些：

特殊字符 代表意义

\*(星号) 代表任何时刻都接受的意思！举例来说，范例一内那个日、月、周都是\* ， 就代表着『不论何月、何日的礼拜几的12:00 都执行后续指令』的意思！

,(逗号) 代表分隔时段的意思。举例来说，如果要下达的工作是3:00 与6:00 时，就会是：

0 3,6 \* \* \* command

时间参数还是有五栏，不过第二栏是3,6 ，代表3 与6 都适用！

-(减号) 代表一段时间范围内，举例来说， 8 点到12 点之间的每小时的20 分都进行一项工作：

20 8-12 \* \* \* command

仔细看到第二栏变成8-12 喔！代表8,9,10,11,12 都适用的意思！

/n(斜线) 那个n 代表数字，亦即是『每隔n 单位间隔』的意思，例如每五分钟进行一次，则：

\*/5 \* \* \* \* command

很简单吧！用\* 与/5 来搭配，也可以写成0-59/5 ，相同意思！

/etc/crontab, /etc/cron.d/\*

系统的例行性任务，只要编辑 /etc/crontab这个档案就可以啦

cron这个服务的最低侦测限制是『分钟』，所以『 cron会每分钟去读取一次/etc/crontab与/var/spool/cron里面的资料内容

/etc/crontab编辑保存退出后最好重启cron服务,systemctl restart crond

[root@study ~]# cat /etc/crontab

SHELL=/bin/bash <==使用哪种shell介面

PATH=/sbin:/bin:/usr/sbin:/usr/bin <==执行档搜寻路径

MAILTO=root <==若有额外STDOUT，以email将资料送给谁

# Example of job definition:

# .---------------- minute (0 - 59)

# | .------------- hour (0 - 23)

# | | .---------- day of month (1 - 31)

# | | | .------- month (1 - 12) OR jan,feb,mar,apr ...

# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat

# | | | | |

# \* \* \* \* \* user-name command to be executed #注意比crontab命令多了一个用户

crond 预设有三个地方会有执行脚本设定档，他们分别是：

/etc/crontab

/etc/cron.d/\* #其它包括每日或每周每月相关的任务

/var/spool/cron/\* #用户的任务

如[root@study ~]# cat /etc/cron.d/0hourly

# Run the hourly jobs

SHELL=/bin/bash

PATH=/sbin:/bin:/usr/sbin:/usr/bin

MAILTO=root

01 \* \* \* \* root run-parts /etc/cron.hourly

可以直接将指令放置到(或连结到)/etc/cron.hourly/ 目录下，则该指令就会被crond 在每小时的1 分开始后的5 分钟内，随机取一个时间点来执行啰！你无须手动去指定分、时、日、月、周就是了,注意放的要是执行文件,其它的类似

注意事项

资源分配不均的问题

如同一分钟执行四个命令,可以错开执行的分钟时间

取消不要的输出项目

利用重定向到 /dev/null

安全的检验

查看/var/log/cron有无木马例行任务

周与日月不可同时并存

可唤醒停机期间的工作任务

anacron 会每个小时被主动执行一次,用来执行Linux关机后并未正常执行cron的任务

crond 会主动去读取/etc/crontab, /var/spool/cron/\*, /etc/cron.d/\* 等设定档，并依据『分、时、日、月、周』的时间设定去各项工作排程；

根据/etc/cron.d/0hourly 的设定，主动去/etc/cron.hourly/ 目录下，执行所有在该目录下的执行档；

因为/etc/cron.hourly/0anacron 这个指令档的缘故，主动的每小时执行anacron ，并呼叫/etc/anacrontab 的设定档；

根据/etc/anacrontab 的设定，依据每天、每周、每月去分析/etc/cron.daily/, /etc/cron.weekly/, /etc/cron.monthly/ 内的执行档，以进行固定周期需要执行的指令。

如果你每个周日的需要执行的动作是放置于/etc/crontab 的话，那么该动作只要过期了就过期了，并不会被调回来重新执行。但如果是放置在/etc/cron.weekly/ 目录下，那么该工作就会定期，几乎一定会在一周内执行一次～如果你关机超过一周，那么一开机后的数个小时内，该工作就会主动的被执行喔

进程管理与SElinux初控

触发任何一个事件时,系统都会将它定义为一个进程,并给予一个ID，PID，同时根据触发这个进程的用户与相关属性,给予这个PID一组有效的权限

程序一般是放在磁盘中,用户通过执行触发后,加载到内存成为一个个体,进程,为了让操作系统管理,进程会有执行者的权限/属性等参数,以及进程需要的脚本与数据,最后再给一个PID，操作系统根据PID来判断进程是否具有执行权限

如BASH，用户登录后执行BASH，有PID，PID根据登录者的UID/GID来,每个用户登录的BASH获取的PID与UID/GID不一样,而当这个进程执行其他作业时,如touch,那由这个进程程衍生出来的其它进程一般状态下也会沿用这个进程的相关权限

[root@localhost ~]# bash #触发BASH

[root@localhost ~]# ps -l

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 0 11408 11402 0 80 0 - 28959 do\_wai pts/1 00:00:00 bash

4 S 0 11439 11408 0 80 0 - 28959 do\_wai pts/1 00:00:00 bash

0 R 0 11454 11439 0 80 0 - 38309 - pts/1 00:00:00 p

#第二个BASH的PPID是第一个BASH的pID，因为第二个BASH是第一个衍生出来的子进程

fork and exec (程序调用的流程)

进程都会借父进程以复制(fork)的方式产生一个一模一样的子进程(临时中间进程)

子进程(临时中间进程)以exec的方式来执行实际要执行的进程,最终成为一个子进程

系统或网络服务: 常驻在内存中的进程

ls等进程执行后就立即结束了,cron等进程启动后会一直在内存中,这叫做常驻在内存中的进程

常驻在内存中的进程都是负责一些系统提供的功能以服务用户的各项任务,因此也称为 服务

系统服务

网络服务,启动后会提供端口port来给外部客户端请求

任务管理(job control)

登录系统获取BASH后,在单一终端下同时执行多个任务的操作管理

可以出现提示符让你操作的环境就称为前台(foregroud),至于其它的任务就可以放入后台(backgroud)去暂停或运行

注意

这些任务所触发的进程必须来自于你的SHELL的子进程(只管理自己的bash)

前台: 可以控制与执行命令的这个环境称为前台的任务(foregroud)

后台: 可以自动执行的任务,你无法使用ctrl + c 终止它,可使用bg, fg调用该任务

后台中执行的进程不能等等terminal或shell的输入,如 vim

BASH只能管理自己的任务不能管理其他的任务,所以即使是root也不能将别人bash下面的job拿过来执行

直接将命令丢到后台执行 &

如tar -zcvpf /tmp/etc.tar.gz /etc &> /tmp/log.txt & #注意数据流重导向写入文件,不然会输出到屏幕

执行后会显示

[1] 13343 <== 等 任务号码与PID

完成后会有显示

[1]+ Done tar -zcf....(执行的命令)

将目前的任务丢到后台中(暂停), ctrl -z

[root@study ~]# vim ~/.bashrc

#在vim的一般模式下，按下[ctrl]-z这两个按键

[1]+ Stopped vim ~/.bashrc

[root@study ~]# <==顺利取得了前景的操控权！

[root@study ~]# find / -print

....(输出省略)....

# 此时萤幕会非常的忙碌！因为萤幕上会显示所有的档名。请按下[ctrl]-z 暂停

[2]+ Stopped find / -print

#+代表最近一个被丢进背景的工作，且目前在背景下预设会被取用的那个工作(与fg这个指令有关)！而那个Stopped则代表目前这个工作的状态。在预设的情况下，使用[ctrl]-z丢到背景当中的工作都是『暂停』的状态喔

[root@study ~]# jobs [-lrs]

选项与参数：

-l ：除了列出job number 与指令串之外，同时列出PID 的号码；

-r ：仅列出正在背景run 的工作；

-s ：仅列出正在背景当中暂停(stop) 的工作。

范例一：观察目前的bash当中，所有的工作，与对应的PID

[root@study ~]# jobs -l

[1]- 14566 Stopped vim ~/.bashrc

[2]+ 14567 Stopped find / -print

目前我有两个工作在背景当中，两个工作都是暂停的，而如果我仅输入fg时，那么那个[2]会被拿到前景当中来处理

其实+代表最近被放到背景的工作号码， -代表最近最后第二个被放置到背景中的工作号码。 而超过最后第三个以后的工作，就不会有+/-符号存在了

将背景工作拿到前景来处理：fg

[root@study ~]# fg %jobnumber

选项与参数：

%jobnumber ：jobnumber 为工作号码(数字)。注意，那个% 是可有可无的！

范例一：先以jobs观察工作，再将工作取出：

[root@study ~]# jobs -l

[1]- 14566 Stopped vim ~/.bashrc

[2]+ 14567 Stopped find / -print

[root@study ~]# fg <==预设取出那个+的工作，亦即[2]。立即按下[ctrl]-z

[root@study ~]# fg %1 <==直接规定取出的那个工作号码！再按下[ctrl]-z

[root@study ~]# jobs -l

[1] + 14566 Stopped vim ~/.bashrc

[2]- 14567 Stopped find / -print

让工作在背景下的状态变成运作中： bg

范例一：一执行find / -perm /7000 > /tmp/text.txt后，立刻丢到背景去暂停！

[root@study ~]# find / -perm /7000 > /tmp/text.txt

#此时，请立刻按下[ctrl]-z暂停！

[3]+ Stopped find / -perm /7000 > /tmp/text.txt

范例二：让该工作在背景下进行，并且观察他！！

[root@study ~]# jobs ; bg %3 ; jobs

[1] Stopped vim ~/.bashrc

[2]- Stopped find / -print

[3]+ Stopped find / -perm /7000 >

/tmp/text.txt

[3]+ find / -perm /7000 > /tmp/text.txt &

[1]- Stopped vim ~/.bashrc

[2]+ Stopped find / -print

[3] Running find / -perm /7000 > /tmp/text.txt &

那个状态列～以经由Stop​​ping 变成了Running 啰！看到差异点，嘿嘿！指令列最后方多了一个& 的符号

管理背景当中的工作： kill

[root@study ~]# kill -signal %jobnumber

[root@study ~]# kill -l

选项与参数：

-l ：这个是L 的小写，列出目前kill 能够使用的讯号(signal) 有哪些？

signal ：代表给予后面接的那个工作什么样的指示啰！用man 7 signal 可知：

-1 ：重新读取一次参数的设定档(类似reload)；

-2 ：代表与由键盘输入[ctrl]-c 同样的动作；

-9 ：立刻强制删除一个工作；

-15：以正常的程序方式终止一项工作。与-9 是不一样的。 #kill的默认值

范例一：找出目前的bash环境下的背景工作，并将该工作『强制删除』。

[root@study ~]# jobs

[1]+ Stopped vim ~/.bashrc

[2] Stopped find / -print

[root@study ~]# kill -9 %2; jobs

[1]+ Stopped vim ~/.bashrc

[2] Killed find / -print

#再过几秒你再下达jobs一次，就会发现2号工作不见了！因为被移除了！

范例二：找出目前的bash环境下的背景工作，并将该工作『正常终止』掉。

[root@study ~]# jobs

[1]+ Stopped vim ~/.bashrc

[root@study ~]# kill -SIGTERM %1

# -SIGTERM与-15是一样的！您可以使用kill -l来查阅！

# 不过在这个案例中， vim 的工作无法被结束喔！因为他无法透过kill 正常终止的意思！

kill 后面接的数字预设会是PID ，如果想要管理bash 的工作控制，就得要加上%数字

离线管理问题

上面的是BASH的后台任务,还是与终端相关,终端关掉后后台执行的任务还是会中断,并不是真正的系统后台

想让终端离线后还能继续执行任务,可以用at与nohup

[root@study ~]# nohup [指令与参数] <==在终端机前景中工作

[root@study ~]# nohup [指令与参数] & <==在终端机背景中工作

# 1.先编辑一支会『睡着500秒』的程式：

[root@study ~]# vim sleep500.sh

#!/bin/bash

/bin/sleep 500s

/bin/echo "I have slept 500 seconds."

# 2.丢到背景中去执行，并且立刻登出系统：

[root@study ~]# chmod a+x sleep500.sh

[root@study ~]# nohup ./sleep500.sh &

[2] 14812

[root@study ~]# nohup: ignoring input and appending output to `nohup.out' <==会告知这个讯息！

[root@study ~]# exit

再次终端连入会发现脚本还是在执行

注意,nohup不支持BASH的内建命令，必须是外部指令

进程管理

仅观察自己的bash 相关程序： ps -l

范例一：将目前属于您自己这次登入的PID与相关资讯列示出来(只与自己的bash有关)

[root@study ~]# ps -l

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 0 14830 13970 0 80 0 - 52686 poll\_s pts/0 00:00:00 sudo

4 S 0 14835 14830 0 80 0 - 50511 wait pts/0 00:00:00 su

4 S 0 14836 14835 0 80 0 - 29035 wait pts/0 00:00:00 bash

0 R 0 15011 14836 0 80 0 - 30319 - pts/0 00:00:00 ps

F：代表这个程序旗标(process flags)，说明这个程序的总结权限，常见号码有：

若为4 表示此程序的权限为root ；

若为1则表示此子程序仅进行复制(fork)而没有实际执行(exec)。

S：代表这个程序的状态(STAT)，主要的状态有：

R (Running)：该程式正在运作中；

S (Sleep)：该程式目前正在睡眠状态(idle)，但可以被唤醒(signal)。

D ：不可被唤醒的睡眠状态，通常这支程式可能在等待I/O 的情况(ex>列印)

T ：停止状态(stop)，可能是在工作控制(背景暂停)或除错(traced) 状态；

Z (Zombie)：僵尸状态，程序已经终止但却无法被移除至记忆体外。

UID/PID/PPID：代表『此程序被该UID 所拥有/程序的PID 号码/此程序的父程序PID 号码』

C：代表CPU 使用率，单位为百分比；

PRI/NI：Priority/Nice的缩写，代表此程序被CPU所执行的优先顺序，数值越小代表该程序越快被CPU执行。详细的PRI与NI将在下一小节说明。

ADDR/SZ/WCHAN：都与记忆体有关，ADDR 是kernel function，指出该程序在记忆体的哪个部分，如果是个running 的程序，一般就会显示『 - 』 / SZ 代表此程序用掉多少记忆体/ WCHAN 表示目前程序是否运作中，同样的， 若为- 表示正在运作中。

TTY：登入者的终端机位置，若为远端登入则使用动态终端介面(pts/n)；

TIME：使用掉的CPU 时间，注意，是此程序实际花费CPU 运作的时间，而不是系统时间；

CMD：就是command 的缩写，造成此程序的触发程式之指令为何

观察系统所有程序： ps aux

范例二：列出目前所有的正在记忆体当中的程序：

[root@study ~]# ps aux

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

root 1 0.0 0.2 60636 7948 ? Ss Aug04 0:01 /usr/lib/systemd/systemd ...

root 2 0.0 0.0 0 0 ? S Aug04 0:00 [kthreadd]

.....(中间省略).....

root 14830 0.0 0.1 210744 3988 pts/0 S Aug04 0:00 sudo su -

root 14835 0.0 0.1 202044 2996 pts/0 S Aug04 0:00 su -

root 14836 0.0 0.1 116140 2960 pts/0 S Aug04 0:00 -bash

.....(中间省略).....

root 18459 0.0 0.0 123372 1380 pts/0 R+ 00:25 0:00 ps aux

你会发现ps -l 与ps aux 显示的项目并不相同！在ps aux 显示的项目中，各栏位的意义为：

USER：该process 属于那个使用者帐号的？

PID ：该process 的程序识别码。

%CPU：该process 使用掉的CPU 资源百分比；

%MEM：该process 所占用的实体记忆体百分比；

VSZ ：该process 使用掉的虚拟记忆体量(Kbytes)

RSS ：该process 占用的固定的记忆体量(Kbytes)

TTY ：该process 是在那个终端机上面运作，若与终端机无关则显示?，另外， tty1-tty6 是本机上面的登入者程序，若为pts/0 等等的，则表示为由网路连接进主机的程序。

STAT：该程序目前的状态，状态显示与ps -l 的S 旗标相同(R/S/T/Z)

START：该process 被触发启动的时间；

TIME ：该process 实际使用CPU 运作的时间。

COMMAND：该程序的实际指令为何？

僵尸(zombie) 』程序是什么？通常，造成僵尸程序的成因是因为该程序应该已经执行完毕，或者是因故应该要终止了， 但是该程序的父程序却无法完整的将该程序结束掉，而造成那个程序一直存在记忆体当中。如果你发现在某个程序的CMD 后面还接上<defunct> 时，就代表该程序是僵尸程序啦，例如：

apache 8683 0.0 0.9 83384 9992 ? Z 14:33 0:00 /usr/sbin/httpd <defunct>

当系统不稳定的时候就容易造成所谓的僵尸程序，可能是因为程式写的不好啦，或者是使用者的操作习惯不良等等所造成。如果你发现系统中很多僵尸程序时，记得啊！要找出该程序的父程序，然后好好的做个追踪，好好的进行主机的环境最佳化啊！看看有什么地方需要改善的，不要只是直接将他kill 掉而已呢

通常僵尸程序都已经无法控管，而直接是交给systemd 这支程式来负责了，偏偏systemd 是系统第一支执行的程式， 他是所有程式的父程式！我们无法杀掉该程式的(杀掉他，系统就死掉了！)，所以啰，如果产生僵尸程序， 而系统过一阵子还没有办法透过核心非经常性的特殊处理来将该程序删除时，那你只好透过reboot 的方式来将该程序抹去了！

top：动态观察程序的变化

相对于ps 是撷取一个时间点的程序状态， top 则可以持续侦测程序运作的状态！使用方式如下：

[root@study ~]# top [-d数字] | top [-bnp]

选项与参数：

-d ：后面可以接秒数，就是整个程序画面更新的秒数。预设是5 秒；

-b ：以批次的方式执行top ，还有更多的参数可以使用喔！

通常会搭配资料流重导向来将批次的结果输出成为档案。

-n ：与-b 搭配，意义是，需要进行几次top 的输出结果。

-p ：指定某些个PID 来进行观察监测而已。

在top 执行过程当中可以使用的按键指令：

? ：显示在top 当中可以输入的按键指令；

P ：以CPU 的使用资源排序显示；

M ：以Memory 的使用资源排序显示；

N ：以PID 来排序喔！

T ：由该Process 使用的CPU 时间累积(TIME+) 排序。

k ：给予某个PID 一个讯号(signal)

r ：给予某个PID 重新制订一个nice 值。

q ：离开top 软体的按键

范例一：每两秒钟更新一次top ，观察整体资讯：

[root@study ~]# top -d 2

top - 00:53:59 up 6:07, 3 users, load average: 0.00, 0.01, 0.05

Tasks : 179 total, 2 running, 177 sleeping, 0 stopped, 0 zombie

%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa , 0.0 hi, 0.0 si, 0.0 st

KiB Mem : 2916388 total, 1839140 free, 353712 used, 723536 buff/cache

KiB Swap: 1048572 total, 1048572 free, 0 used . 2318680 avail Mem

<==如果加入k或r时，就会有相关的字样出现在这里喔！

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

18804 root 20 0 130028 1872 1276 R 0.5 0.1 0:00.02 top

1 root 20 0 60636 7948 2656 S 0.0 0.3 0:01.70 systemd

2 root 20 0 0 0 0 S 0.0 0.0 0:00.01 kthreadd

3 root 20 0 0 0 0 S 0.0 0.0 0:00.00 ksoftirqd/0

第一行(top...)：这一行显示的资讯分别为：

目前的时间，亦即是00:53:59 那个项目；

开机到目前为止所经过的时间，亦即是up 6:07, 那个项目；

已经登入系统的使用者人数，亦即是3 users, 项目；

系统在1, 5, 15分钟的平均工作负载。我们在第十五章谈到的batch 工作方式为负载小于0.8就是这个负载啰！代表的是1, 5, 15分钟，系统平均要负责运作几个程序(工作)的意思。越小代表系统越闲置，若高于1得要注意你的系统程序是否太过繁复了！

第二行(Tasks...)：显示的是目前程序的总量与个别程序在什么状态(running, sleeping, stopped, zombie)。比较需要注意的是最后的zombie 那个数值，如果不是0 ！好好看看到底是那个process 变成僵尸了吧？

第三行(%Cpus...)：显示的是CPU 的整体负载，每个项目可使用? 查阅。需要特别注意的是wa 项目，那个项目代表的是I/O wait， 通常你的系统会变慢都是I/O 产生的问题比较大！因此这里得要注意这个项目耗用CPU 的资源喔！另外，如果是多核心的设备，可以按下数字键『1』来切换成不同CPU 的负载率。

第四行与第五行：表示目前的实体记忆体与虚拟记忆体(Mem/Swap) 的使用情况。再次重申，要注意的是swap 的使用量要尽量的少！如果swap 被用的很大量，表示系统的实体记忆体实在不足！

第六行：这个是当在top 程式当中输入指令时，显示状态的地方。

至于top 下半部分的画面，则是每个process 使用的资源情况。比较需要注意的是：

PID ：每个process 的ID 啦！

USER：该process 所属的使用者；

PR ：Priority 的简写，程序的优先执行顺序，越小越早被执行；

NI ：Nice 的简写，与Priority 有关，也是越小越早被执行；

%CPU：CPU 的使用率；

%MEM：记忆体的使用率；

TIME+：CPU 使用时间的累加；

范例二：将top的资讯进行2次，然后将结果输出到/tmp/top.txt

[root@study ~]# top -b -n 2 > /tmp/top.txt

#这样一来，嘿嘿！就可以将top的资讯存到/tmp/top.txt档案中了。

范例三：我们自己的bash PID可由$$变数取得，请使用top持续观察该PID

[root@study ~]# echo $$

14836 <==就是这个数字！他是我们bash的PID

[root@study ~]# top -d 2 -p 14836

top - 01:00:53 up 6:14, 3 users, load average: 0.00, 0.01, 0.05

Tasks: 1 total, 0 running, 1 sleeping, 0 stopped, 0 zombie

%Cpu(s): 0.0 us, 0.1 sy, 0.0 ni, 99.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem : 2916388 total, 1839264 free, 353424 used, 723700 buff/cache

KiB Swap: 1048572 total, 1048572 free, 0 used. 2318848 avail Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

14836 root 20 0 116272 3136 1848 S 0.0 0.1 0:00.07 bash

范例四：承上题，上面的NI值是0 ，想要改成10的话？

#在范例三的top画面当中直接按下r之后，会出现如下的图样！

top - 01:02:01 up 6:15, 3 users, load average: 0.00, 0.01, 0.05

Tasks: 1 total, 0 running, 1 sleeping, 0 stopped, 0 zombie

%Cpu(s): 0.1 us, 0.0 sy, 0.0 ni, 99.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem : 2916388 total, 1839140 free, 353576 used, 723672 buff/cache

KiB Swap: 1048572 total, 1048572 free, 0 used. 2318724 avail Mem

PID to renice [default pid = 14836] 14836

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

14836 root 20 0 116272 3136 1848 S 0.0 0.1 0:00.07 bash

在你完成上面的动作后，在状态列会出现如下的资讯：

Renice PID 14836 to value 10 <==这是nice值

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

pstree 可以用来查看进程树,查看父进程

进程的管理

进程如何互相管理?其实是通过给予该进程一个信号(signal),去告知该进程你想要让它做什么

代号 名称 内容

1 SIGHUP 启动被终止的程序，可让该PID 重新读取自己的设定档，类似重新启动

2 SIGINT 相当于用键盘输入[ctrl]-c 来中断一个程序的进行

9 SIGKILL 代表强制中断一个程序的进行，如果该程序进行到一半， 那么尚未完成的部分可能会有『半产品』产生，类似vim会有.filename.swp 保留下来。

15 SIGTERM 以正常的结束程序来终止该程序。由于是正常的终止， 所以后续的动作会将他完成。不过，如果该程序已经发生问题，就是无法使用正常的方法终止时， 输入这个signal 也是没有用的。

19 SIGSTOP 相当于用键盘输入[ctrl]-z 来暂停一个程序的进行

#只需记 1 9 15即可

通过kill或killall发送信号给进程

kill -signal PID

注意kill后接的数字是PID，KILL后面如果是%number则是后台任务,要区别

以ps 找出rsyslogd 这个程序的PID 后，再使用kill 传送讯息，使得rsyslogd 可以重新读取设定档。

答：

由于需要重新读取设定档，因此signal 是1 号。至于找出rsyslogd 的PID 可以是这样做：

ps aux | grep 'rsyslogd' | grep -v 'grep'| awk '{print $2}'

接下来则是实际使用kill -1 PID，因此，整串指令会是这样：

kill -SIGHUP $(ps aux | grep 'rsyslogd' | grep -v 'grep'| awk '{print $2}')

如果要确认有没有重新启动syslog ，可以参考登录档的内容，使用如下指令查阅：

tail -5 /var/log/messages

如果你有看到类似『Aug 5 01:25:02 study rsyslogd: [origin software="rsyslogd" swVersion="7.4.7" x-pid="742" x-info="http://www.rsyslog .com"] rsyslogd was HUPed』之类的字样，就是表示rsyslogd 在8/5 有重新启动(restart) 过了！

killall -signal 命令名称

[root@study ~]# killall [-iIe] [command name]

选项与参数：

-i ：interactive 的意思，互动式的，若需要删除时，会出现提示字元给使用者；

-e ：exact 的意思，表示『后面接的command name 要一致』，但整个完整的指令

不能超过15 个字元。

-I ：指令名称(可能含参数)忽略大小写。

范例一：给予rsyslogd这个指令启动的PID一个SIGHUP的讯号

[root@study ~]# killall -1 rsyslogd

#如果用ps aux仔细看一下，若包含所有参数，则/usr/sbin/rsyslogd -n才是最完整的！

范例二：强制终止所有以httpd启动的程序(其实并没有此程序在系统内)

[root@study ~]# killall -9 httpd

范例三：依次询问每个bash程式是否需要被终止运作！

[root@study ~]# killall -i -9 bash

Signal bash(13888) ? (y/N) n <==这个不杀！

Signal bash(13928) ? (y/N) n <==这个不杀！

Signal bash(13970) ? (y/N) n <==这个不杀！

Signal bash(14836) ? (y/N) y <==这个杀掉！

#具有互动的功能！可以询问你是否要删除bash这个程式。要注意，若没有-i的参数，

# 所有的bash 都会被这个root 给杀掉！包括root 自己的bash 喔！^\_^

进程的执行顺序

进程的执行顺序与进程的优先级(Priority)与CPU的调度有关

CPU调度

指的是每个进程被CPU运行的规则,cpu调度与操作系统有关

linux系统会给予进程一个所谓的优先级(priority,PRI),PRI值越低代表越优先,PRI值是由内核动态调整的,用户无法直接整PRI值

[root@study ~]# ps -l

FS UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 0 14836 14835 0 90 10 - 29068 wait pts/0 00:00:00 bash

0 R 0 19848 14836 0 90 10 - 30319 - pts/0 00:00:00 ps

#你应该要好奇，怎么我的NI已经是10了？还记得刚刚top的测试吗？我们在那边就有改过一次喔！

由于PRI 是核心动态调整的，我们使用者也无权去干涉PRI ！那如果你想要调整程序的优先执行序时，就得要透过Nice 值了！Nice 值就是上表的NI 啦！一般来说， PRI 与NI 的相关性如下：

PRI(new) = PRI(old) + nice

所以，当nice值为负值时，那么该程序就会降低PRI值，亦即会变的较优先被处理。此外，你必须要留意到：

nice 值可调整的范围为-20 ~ 19 ；

root 可随意调整自己或他人程序的Nice 值，且范围为-20 ~ 19 ；

一般使用者仅可调整自己程序的Nice 值，且范围仅为0 ~ 19 (避免一般用户抢占系统资源)；

一般使用者仅可将nice 值越调越高，例如本来nice 为5 ，则未来仅能调整到大于5；

这也就是说，要调整某个程序的优先执行序，就是『调整该程序的nice 值』啦！那么如何给予某个程序nice 值呢？有两种方式，分别是：

一开始执行程式就立即给予一个特定的nice 值：用nice 指令；

调整某个已经存在的PID 的nice 值：用renice 指令。

nice ：新执行的指令即给予新的nice 值

[root@study ~]# nice [-n数字] command

选项与参数：

-n ：后面接一个数值，让原本的nice 加上这个新的数值之意。修改后的最终数值的范围则为-20 ~ 19。

范例一：用root让原本的nice再减少5 (-5)，用于执行vim ，并观察该程序！

[root@study ~]# nice -n -5 vim &

[1] 19865

[root@study ~]# ps -l

FS UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 0 14836 14835 0 90 10 - 29068 wait pts/0 00:00:00 bash

4 T 0 19865 14836 0 85 5 - 37757 signal pts/0 00:00:00 vim

0 R 0 19866 14836 0 90 10 - 30319 - pts/0 00:00:00 ps

# 原本的bash PRI 为90 ，所以vim 预设应为90。不过由于给予nice 为-5 ，

# 因此vim 的PRI 降低了！RPI 与NI 各减5 ！但不一定每次都是正好相同喔！因为核心会动态调整

[root@study ~]# kill -9 %1 <==测试完毕将vim关闭

通常什么时候要将nice 值调大呢？举例来说，系统的背景工作中， 某些比较不重要的程序之进行：例如备份工作！由于备份工作相当的耗系统资源， 这个时候就可以将备份的指令之nice 值调大一些，可以使系统的资源分配的更为公平！

renice ：已存在进程的nice 重新调整

[root@study ~]# renice [number] PID

选项与参数：

PID ：某个程序的ID 啊！

范例一：找出自己的bash PID ，并将该PID的nice调整到-5

[root@study ~]# ps -l

FS UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 0 14836 14835 0 90 10 - 29068 wait pts/0 00:00:00 bash

0 R 0 19900 14836 0 90 10 - 30319 - pts/0 00:00:00 ps

[root@study ~]# renice -5 14836

14836 (process ID) old priority 10, new priority -5

[root@study ~]# ps -l

FS UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 0 14836 14835 0 75 -5 - 29068 wait pts/0 00:00:00 bash

0 R 0 19910 14836 0 75 -5 - 30319 - pts/0 00:00:00 ps

如果要调整的是已经存在的某个程序的话，那么就得要使用renice 了。使用的方法很简单， renice 后面接上数值及PID 即可。因为后面接的是PID ，所以你务必要以ps 或者其他程序观察的指令去找出PID 才行啊！

由上面这个范例当中我们也看的出来，虽然修改的是bash那个程序，但是该程序所触发的ps指令当中的nice也会继承而为-5喔！了解了吧！整个nice值是可以在父程序-->子程序之间传递的呢！另外，除了renice之外，其实那个top同样的也是可以调整nice值的！

查看系统资源信息

free 查看内存使用情况

[root@study ~]# free [-b|-k|-m|-g|-h] [-t] [-s N -c N]

选项与参数：

-b ：直接输入free 时，显示的单位是Kbytes，我们可以使用b(bytes), m(Mbytes)

k(Kbytes), 及g(Gbytes) 来显示单位喔！也可以直接让系统自己指定单位(-h)

-t ：在输出的最终结果，显示实体记忆体与swap 的总量。

-s ：可以让系统每几秒钟输出一次，不间断的一直输出的意思！对于系统观察挺有效！

-c ：与-s 同时处理～让free 列出几次的意思～

范例一：显示目前系统的记忆体容量

[root@study ~]# free -m

total used free shared buff/cache available

Mem: 2848 346 1794 8 706 2263

Swap: 1023 0 1023

#free剩余可用量,shared buff/cache则是在已被已使用的量当中,用来作为缓冲及缓存的,这些使用量中,在系统比较忙时,可以发布而继续利用,因此后面有一个availabe

如这机子本身没什么服务,但却有706M的缓存,这些文件被系统暂缓存下来,等待下次运行时可以更快速地取出,也就是说系统是很有效率地将所有的内存用光,目的是让系统的读写性能加速

swap量一般来说不要被使用,如果swap使用超过20%,最好增加物理内存,因为swap跟物理内存的性能差太多

[root@study ~]# uname [-asrmpi]

选项与参数：

-a ：所有系统相关的资讯，包括底下的资料都会被列出来；

-s ：系统核心名称

-r ：核心的版本

-m ：本系统的硬体名称，例如i686 或x86\_64 等；

-p ：CPU 的类型，与-m 类似，只是显示的是CPU 的类型！

-i ：硬体的平台(ix86)

范例一：输出系统的基本资讯

[root@study ~]# uname -a

Linux study.centos.vbird 3.10.0-229.el7.x86\_64 #1 SMP Fri Mar 6 11:36:42 UTC 2015

x86\_64 x86\_64 x86\_64 GNU/Linux

Linux 主机使用的核心名称为Linux，而主机名称为study.centos.vbird，核心的版本为3.10.0-229.el7.x86\_64 ，该核心版本建立的日期为2015-3-6，适用的硬体平台为x86\_64 以上等级的硬体平台喔。

[root@study ~]# uptime #查看cpu平均任务负载

02:35:27 up 7:48, 3 users, load average: 0.00, 0.01, 0.05

[root@study ~]# netstat -[atunlp]

选项与参数：

-a ：将目前系统上所有的连线、监听、Socket 资料都列出来

-t ：列出tcp 网路封包的资料

-u ：列出udp 网路封包的资料

-n ：不以程序的服务名称，以埠号(port number) 来显示；

-l ：列出目前正在网路监听(listen) 的服务；

-p ：列出该网路服务的程序PID

范例一：列出目前系统已经建立的网路连线与unix socket状态

[root@study ~]# netstat

Active Internet connections (w/o servers) <==与网路较相关的部分

Proto Recv-Q Send-Q Local Address Foreign Address State

tcp 0 0 172.16.15.100:ssh 172.16.220.234:48300 ESTABLISHED

Active UNIX domain sockets (w/o servers) <==与本机的程序自己的相关性(非网路)

Proto RefCnt Flags Type State I-Node Path

unix 2 [ ] DGRAM 1902 @/org/freedesktop/systemd1/notify

unix 2 [ ] DGRAM 1944 /run/systemd/shutdownd

....(中间省略)....

unix 3 [ ] STREAM CONNECTED 25425 @/tmp/.X11-unix/X0

unix 3 [ ] STREAM CONNECTED 28893

unix 3 [ ] STREAM CONNECTED 21262

范例二：找出目前系统上已在监听的网路连线及其PID

[root@study ~]# netstat -tulnp

Active Internet connections (only servers)

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN 1326/sshd

tcp 0 0 127.0.0.1:25 0.0.0.0:\* LISTEN 2349/master

tcp6 0 0 :::22 :::\* LISTEN 1326/sshd

tcp6 0 0 ::1:25 :::\* LISTEN 2349/master

udp 0 0 0.0.0.0:123 0.0.0.0:\* 751/chronyd

udp 0 0 127.0.0.1:323 0.0.0.0:\* 751/chronyd

udp 0 0 0.0.0.0:57808 0.0.0.0:\* 743/avahi-daemon: r

udp 0 0 0.0.0.0:5353 0.0.0.0:\* 743/avahi-daemon: r

udp6 0 0 :::123 :::\* 751/chronyd

udp6 0 0 ::1:323 :::\* 751/chronyd

# 除了可以列出监听网路的介面与状态之外，最后一个栏位还能够显示此服务的

# PID 号码以及程序的指令名称喔！例如上头的1326 就是该PID

范例三：将上述的0.0.0.0:57808那个网路服务关闭的话？

[root@study ~]# kill -9 743

[root@study ~]# killall -9 avahi-daemon

范例一：输出所有的核心开机时的资讯

[root@study ~]# dmesg | more

范例二：搜寻开机的时候，硬碟的相关资讯为何？

[root@study ~]# dmesg | grep -i vda

[ 0.758551] vda: vda1 vda2 vda3 vda4 vda5 vda6 vda7 vda8 vda9

[ 3.964134] XFS (vda2): Mounting V4 Filesystem

vmstat ：侦测系统资源变化

[root@study ~]# vmstat [-a] [延迟[总计侦测次数]] <==CPU/记忆体等资讯

[root@study ~]# vmstat [-fs] <==记忆体相关

[root @study ~]# vmstat [-S单位] <==设定显示数据的单位

[root@study ~]# vmstat [-d] <==与磁碟有关

[root@study ~]# vmstat [-p分割槽] <==与磁碟有关

选项与参数：

-a ：使用inactive/active(活跃与否) 取代buffer/cache 的记忆体输出资讯；

-f ：开机到目前为止，系统复制(fork) 的程序数；

-s ：将一些事件(开机至目前为止) 导致的记忆体变化情况列表说明；

-S ：后面可以接单位，让显示的资料有单位。例如K/M 取代bytes 的容量；

-d ：列出磁碟的读写总量统计表

-p ：后面列出分割槽，可显示该分割槽的读写总量统计表

范例一：统计目前主机CPU状态，每秒一次，共计三次！

[root@study ~]# vmstat 1 3

procs ------------memory---------- ---swap-- -----io---- -system-- ------ cpu-----

rb swpd free buff cache si so bi bo in cs us sy id wa st

1 0 0 1838092 1504 722216 0 0 4 1 6 9 0 0 100 0 0

0 0 0 1838092 1504 722200 0 0 0 0 13 23 0 0 100 0 0

0 0 0 1838092 1504 722200 0 0 0 0 25 46 0 0 100 0 0

程序栏位(procs)的项目分别为：

r ：等待运作中的程序数量；b：不可被唤醒的程序数量。这两个项目越多，代表系统越忙碌(因为系统太忙，所以很多程序就无法被执行或一直在等待而无法被唤醒之故)。

记忆体栏位(memory)项目分别为：

swpd：虚拟记忆体被使用的容量； free：未被使用的记忆体容量； buff：用于缓冲记忆体； cache：用于快取记忆体。这部份则与free是相同的。

记忆体置换空间(swap)的项目分别为：

si：由磁碟中将程序取出的量； so：由于记忆体不足而将没用到的程序写入到磁碟的swap的容量。如果si/so的数值太大，表示记忆体内的资料常常得在磁碟与主记忆体之间传来传去，系统效能会很差！

磁碟读写(io)的项目分别为：

bi：由磁碟读入的区块数量； bo：写入到磁碟去的区块数量。如果这部份的值越高，代表系统的I/O非常忙碌！

系统(system)的项目分别为：

in：每秒被中断的程序次数； cs：每秒钟进行的事件切换次数；这两个数值越大，代表系统与周边设备的沟通非常频繁！这些周边设备当然包括磁碟、网路卡、时间钟等。

CPU的项目分别为：

us：非核心层的CPU使用状态； sy：核心层所使用的CPU状态； id：闲置的状态； wa：等待I/O所耗费的CPU状态(达%3以上磁盘有瓶劲)； st：被虚拟机器( virtual machine)所盗用的CPU使用状态(2.6.11以后才支援)

CPU使用率可以这样查看 vmstat | awk 'NR==3{print $13+$14}'

特殊文件与进程

suid的权限与进程的相关性非常大

如一般用户执行passwd后就具有了root的权限(暂时),这是因为触发passwd后,会获得一个新的进程与PID,该PID产生时通过SUID来给予该PID特珠的权限设置

[dmtsai@study ~]$ passwd

Changing password for user dmtsai.

Changing password for dmtsai

(current) UNIX password: <==这里按下[ctrl]-z并且按下[enter]

[1]+ Stopped passwd

[dmtsai@study ~]$ pstree -uA

systemd-+-ModemManager---2\*[{ModemManager}]

....(中间省略)....

|-sshd---sshd--- sshd(dmtsai)---bash -+- passwd(root)

| `-pstree

#可以发现,passwd确实是dmtsai的bash衍生出来的,不过权限不一样(root权限)

/proc/\* 代表的意义

进程都是在内存中的,而内存当中的数据又都是写入/proc/\*里面,主机上面的PID都以目录的形式存在于/proc当中,了解下即可

fuser

借由文件或(文件系统)找出正在使用该文件的进程

如fuser -uv .

cd /home

fuser -muv .

USER PID ACCESS COMMAND

/home: root kernel mount (root)/home

root 20560 ..c.. (root)bash

#通过上面可知有进程在home目录下运行,所以即使离开了/home目录,也是不能umout，只能通过fuser -mki /home 删除进程才能umount

lsof

与fuser刚好相反,是列出被进程所使用的文件名称

lsof -u root | grep 'bash'

#列出用户root的bash进程所开户使用的文件

pidof

找出某个正在执行的进程的PID

pidof systemd rsyslogd

1 3063

selinux

Security Enhanced Linux,安全强化Linux的意思

最美国国家安全局(NSA)开发,当初开发的原因是很多企业发现,系统出现问题的原因大部分在于(内部员工的资源误用),实际由外部发动的攻击反而没那么严重

资源误用,举例来说如果有个不是很懂系统的系统管理员为了自己设置方便,将网站网录/var/www/html的权限设置为777,这样的话所有的进程都可对该目录读写，只要该服务启动后又是面向的是internet,有心人接触到这个进程,就会乱写东西进去

selinux是在进行进程,文件等详细权限配置时依据的一个内核模块,由于启动网络服务也是进程,因此也能控制网络服务能否读写资源的一道关卡

传统的文件权限与账号的关系: 自主访问控制(DAC)

系统的账号分为root与一般用户,root拥用所有的权限,当某个进程想要对文件进行读写时,系统就会根据该进程的拥有者与用户组,比对文件的权限,权限通过才可以读写文件

缺点

root具有最高的权限,如果某进程被有心人获取,肯该进程属于root，那该进程就可以在系统上对任何资源读写

如某个目录不小心设置为777,那么该目录任何人都可以读写

以规则策略制定特定进程读取特定的文件: 强制访问控制MAC

针对特定的进程与特定的文件资源来管理,如即使是root,那么在使用不同进程时,能获取的权限也不一定是root,针对控制的主体变成了进程而不是用户，进程也不能任意使用系统资源,因为每个系统资源也也会针对该主体进程设置了可使用的权限

如httpd这个进程仅能在/var/www这个目录下读写,进程想要去其它目录读写还需要开放相应的权限才行

selinux 运行模式

主体(subject)

selinux管理的是进程,主体与进程划上等号

目标(object)

主体进程能否读写的目标资源(文件系统)

策略(policy)

由于进程与文件数量庞大,因此selinux会依据某些服务来订制基本的读写安全性策略,策略还会有详细的规则来指定不同的服务是否开放某些资源的读写，目前centos 7 里面仅提供三个主要的策略

targeted 针对网络服务限制多,针对本机限制较少,默认策略

minimum 由target自定义而来,仅针对选择的进程来保护

mis 完整的selinux限制,较为严格

建议使用默认的targeted就好

主本(subject) --(请求)> selinux --> 安全上下文--(yes)->目标

| | |

分析策略 no |

| | 能否存取还

策略 拒绝 参考文件的

rule1 rwx

rule2

可看主体如何获取目录,1）必须要先通过selinux策略的规则放行后,才可以与目标资源进行安全上下文的比对 2)比对失败则不法读写目标，苦比对成功则可以读写,最终能否读定还与文件的rwx有关

安全上下文

targetet的策略已经帮订好非常多的规则,我们只要知道如何开启/关闭即可,但安全上下文可能要自行配置,可以将安全上下文当作selinux内必备的rwx

安全上下文存在于主体进程与文件资源中,进程在内存中,所以安全上下文存入没问题.而文件的安全上下文记录则是放到文件的inode内,因此进程想要读取文件资源,也要读取inode,在Inode内比对安全上下文与rwx是否正确而给予适当的权限

查看安全上下文 ls -Z （selinux要开启

#先来观察一下root家目录底下的『文件的SELinux相关资讯』

[root@study ~]# ls -Z

-rw-------. root root system\_u:object\_r:admin\_home\_t:s0 anaconda-ks. cfg

-rw-r--r--. root root system\_u:object\_r:admin\_home\_t:s0 initial-setup-ks.cfg

-rw-r--r--. root root unconfined\_u:object\_r:admin\_home\_t:s0 regular\_express.txt

#上述特殊字体的部分，就是安全上下文的内容！鸟哥仅列出数个预设的文件而已，

# 本书学习过程中所写下的文件则没有列在上头喔！

如上所示，安全性本文主要用冒号分为三个栏位，这三个栏位的意义为：

Identify:role:type

身份识别:角色:类型

这三个栏位的意义仔细的说明一下吧：

身份识别(Identify)：

相当于帐号方面的身份识别！主要的身份识别常见有底下几种常见的类型：

unconfined\_u：不受限的用户，也就是说，该文件来自于不受限的进程所产生的！一般来说，我们使用可登入帐号来取得bash之后，预设的bash环境是不受SELinux管制的～因为bash并不是什么特别的网路服务！因此，在这个不受SELinux所限制的bash进程所产生的文件，其身份识别大多就是unconfined\_u这个『不受限』用户啰！

system\_u：系统用户，大部分就是系统自己产生的文件啰！

基本上，如果是系统或软体本身所提供的文件，大多就是system\_u 这个身份名称，而如果是我们用户透过bash 自己建立的文件，大多则是不受限的unconfined\_u 身份～如果是网路服务所产生的文件，或者是系统服务运作过程产生的文件，则大部分的识别就会是system\_u 啰！

因为鸟哥这边教大家使用文字界面来产生许多的资料，因此你看上面的三个文件中，系统安装主动产生的anaconda-ks.cfs 及initial-setup-ks.cfg 就会是system\_u，而我们自己从网路上面抓下来的regular\_express.txt 就会是unconfined\_u 这个识别啊！

角色(Role)：

透过角色栏位，我们可以知道这个资料是属于进程、文件资源还是代表使用者。一般的角色有：

object\_r：代表的是文件或目录等文件资源，这应该是最常见的啰；

system\_r：代表的就是进程啦！不过，一般使用者也会被指定成为system\_r喔！

你也会发现角色的栏位最后面使用『 \_r 』来结尾！因为是role 的意思嘛！

类型(Type) (最重要！)：

在预设的targeted 政策中， Identify 与Role 栏位基本上是不重要的！重要的在于这个类型(type) 栏位！基本上，一个主体进程能不能读取到这个文件资源，与类型栏位有关！而类型栏位在文件与进程的定义不太相同，分别是：

type：在文件资源(Object) 上面称为类型(Type)；

domain：在主体进程(Subject) 则称为域(domain) 了！

domain 需要与type 搭配，则该进程才能够顺利的读取文件资源啦！

进程与文件selinux类型字段的相关性

再来观察一下系统『进程的SELinux相关资讯』

[root@study ~]# ps -eZ

LABEL PID TTY TIME CMD

system\_u:system\_r:init\_t:s0 1 ? 00:00:03 systemd

system\_u:system\_r:kernel\_t:s0 2 ? 00:00:00 kthreadd

system\_u:system\_r :kernel\_t:s0 3 ? 00:00:00 ksoftirqd/0

.....(中间省略).....

unconfined\_u:unconfined\_r:unconfined\_t:s0-s0:c0.c1023 31513 ? 00:00:00 sshd

unconfined\_u:unconfined\_r :unconfined\_t:s0-s0:c0.c1023 31535 pts/0 00:00:00 bash

#基本上进程主要就分为两大类，一种是系统有受限的system\_u:system\_r，另一种则可能是用户自己的，

# 比较不受限的程序(通常是本机用户自己执行的进程)，亦即是unconfined\_u:unconfined\_r 这两种！

身份识别 角色 该对应在targeted 的意义

unconfined\_u unconfined\_r 一般可登入用户的进程啰！比较没有受限的进程之意！大多数都是用户已经顺利登入系统(不论是网路还是本机登入来取得可用的shell) 后， 所用来操作系统的进程！如bash, X window 相关软体等。

system\_u system\_r 由于为系统帐号，因此是非交谈式的系统运作程序，大多数的系统进程均是这种类型！

#在target策略下,其实最重要的字段就是类型字段(type),主体与目标之间是否具有可以读写的权限,与进程的domain及文件的type有关

# 1.先看看crond这个『进程』的安全上下文内容：

[root@study ~]# ps -eZ | grep cron

system\_u:system\_r: crond\_t :s0-s0:c0.c1023 1338 ? 00:00:01 crond

system\_u:system\_r:crond\_t:s0-s0:c0.c1023 1340 ? 00:00:00 atd

# 这个安全本文的类型名称为crond\_t 格式！

# 2.再来瞧瞧执行文件、配置文件等等的安全本上下文内容是什么！

[root@study ~]# ll -Zd /usr/sbin/crond /etc/crontab /etc/cron.d

drwxr-xr-x. root root system\_u:object\_r: system\_cron\_spool\_t :s0 /etc/cron.d

-rw-r--r--. root root system\_u:object\_r: system\_cron\_spool\_t :s0 /etc/crontab

-rwxr-xr-x. root root system\_u:object\_r: crond\_exec\_t :s0 /usr/sbin/crond

1) 首先我们触发一个可执行的目标文件,即具有crond\_exex这个类型的/usr/sbin/crond文件

2）该文件的类型会让这个文件所造成的主体进程(subject)具有crond这个域(domain),我们的策略针对这个域已经制定了许多规则,其中包括这个域可以读取的目标资源类型

3）由于crond domain被设置为可以读取system\_cron\_spool\_t这个类型的目标文件(object),因此你的配置文件放到/etc/cron.d目录下就能够被crond那个进程读取

4）最终能不能读到正确的数据,还是要看rwx是否符合linux的权限

selinux 3种模式的启动 关闭与查看

Enfofcing 强制模式,代表selinux运行中,且已经正确开始限制domain/type

Permissive 宽容模式,代表SElinux运行中,不过仅会有警告信息并不会实际限制domain/type的读写,此模式可用来debug

Disabled 关闭模式,selinux并没有实际运行

并不是所有的进程都会被selinux所管制,通过ps -eZ可查看进程有没有爱限(confined)

找一找crond 与bash 这两只进程是否有被限制吧？

[root@study ~]# ps -eZ | grep -E 'cron|bash'

system\_u:system\_r: crond\_t :s0-s0:c0.c1023 1340 ? 00:00:00 atd

unconfined\_u:unconfined\_r: unconfined\_t :s0-s0:c0.c1023 13888 tty2 00:00:00 bash

unconfined\_u:unconfined\_r: unconfined\_t :s0-s0:c0.c1023 28054 pts/0 00:00:00 bash

unconfined\_u:unconfined\_r: unconfined\_t :s0-s0:c0.c1023 28094 pts/0 00:00:00 bash

system\_u:system\_r: crond\_t :s0-s0:c0.c1023 28174 ? 00:00:00 crond

crond确实是受限的主体进程,而bash因为是本机进程,就是不爱限(unconfined)的类型,也就是说bash是直接去比对rwx

查看目前的selinux模式

[root@localhost ~]# getenforce

Enforcing

查看selinux的策略(policy)

[root@study ~]# sestatus [-vb]

选项与参数：

-v ：检查列于/etc/sestatus.conf 内的文件与进程的安全性上下文内容；

-b ：将目前策略的规则布林值列出，亦即某些规则(rule) 是否要启动(0/1) 之意；

范例一：列出目前的SELinux使用哪个策略(Policy)？

[root@study ~]# sestatus

SELinux status: enabled <==是否启动SELinux

SELinuxfs mount: /sys/fs/selinux <==SELinux的相关文件挂载点

SELinux root directory: /etc/selinux <==SELinux的根目录所在

Loaded policy name: targeted <==目前的策略是什么？

Current mode: enforcing <==目前的模式

Mode from config file: enforcing <==目前配置文件规范的SELinux模式

Policy MLS status: enabled <==是否含有MLS的模式机制

Policy deny\_unknown status: allowed <==是否预设抵挡未知的主体进程

Max kernel policy version: 28

selinux的配置文件

[root@study ~]# vim /etc/selinux/config

SELINUX=enforcing <==调整enforcing|disabled|permissive

SELINUXTYPE=targeted <==目前仅有targeted, mls, minimum三种策略

#可直接修改

selinux的启动与关闭

如果修改策略,需要重新启动,如果Enforcing或Permissive改成Disabled或由Disabled改成其它两个,也必须要重启,这是因为selinux是整合到内核中,只可以在SElinux运行下切换成为强制宽容模式,不能直接关闭selinux

如果由Disabled转到记动selinux模式如Enforcing,由于系统必须要针对文件写入安全上下文的信息,因为启动过程会花不少时间在等待重新写入,而且写完之后还要再重启一次,要花一些时间

selinux在Enforcing与Permissive之间切换的方法

[root@study ~]# setenforce [0|1]

选项与参数：

0 ：转成permissive 宽容模式；

1 ：转成Enforcing 强制模式

范例一：将SELinux在Enforcing与permissive之间切换与观察

[root@study ~]# setenforce 0

[root@study ~]# getenforce

Permissive

[root@study ~]# setenforce 1

[root@study ~]# getenforce

Enforcing

如从Disabled切换成Enforcing之后有一堆服务无法顺利启动(提示/lib/xxx无权限),大多是因为重新写入SElinux类型出错,使用Permissive就没有这个错误,可以在permissive状态下 restorecon -Rv重新还原所有的selinux类型

selinux策略内的规则管理

SELinux 各个规则的布林值查询getsebool

如果想要查询系统上面全部规则的启动与否(on/off，亦即布林值)，很简单的透过sestatus -b 或getsebool -a 均可！

[root@study ~]# getsebool [-a] [规则的名称]

选项与参数：

-a ：列出目前系统上面的所有SELinux 规则的布林值为开启或关闭值

范例一：查询本系统内所有的布林值设定状况

[root@study ~]# getsebool -a

abrt\_anon\_write --> off

abrt\_handle\_event --> off

....(中间省略)....

cron\_can\_relabel --> off #这个跟cornd比较有关！

#了解就行

修改安全上下文

使用restorecon 让档案恢复正确的SELinux type

[root@study ~]# restorecon [-Rv]档案或目录

选项与参数：

-R ：连同次目录一起修改；

-v ：将过程显示到萤幕上

范例三：将/etc/cron.d/底下的档案通通恢复成预设的SELinux type！

[root@study ~]# restorecon -Rv /etc/cron.d

restorecon reset /etc/cron.d/checktime context system\_u:object\_r: shadow\_t :s0->

system\_u:object\_r: system\_cron\_spool\_t :s0

#上面这两行其实是同一行喔！表示将checktime由shadow\_t改为system\_cron\_spool\_t

范例四：重新启动crond看看有没有正确启动checktime啰！？

[root@study ~]# systemctl restart crond

[root@study ~]# tail /var/log/cron

#再去瞧瞧这个/var/log/cron的内容，应该就没有错误讯息了

#服务权限或运行不了,用restorecon即可, selinux一般都是不开，略过selinux学习

认识系统服务

daemon与服务

常驻在内存的进程一般称为服务,服务提供系统或网络功能如系统周期性任务(cron),在内存的服务(service)就是由crond来提供守望护的,daemon就等于serivce

服务的进程(daemon)都会以d结尾,如crond,httpd等

早期的system V 的init管理

服务的启动与关闭都是在/etc/init.d目录下

/etc/init.d/daemon start/restart/stop/status

服务启动的分类

单独的服务

超级守望护进程

xined总程序统一管理启动唤醒,缺点唤醒有一点实际的延迟

服务的依赖

运行级别的分类等相关

启动的运行级别

chkconfig daemon on/of

(插入学习)

rsync

rsync -avz --delete /src/abc/ /det

#/src/abc没有/,会把abc目录内空包括abc目录全部复制到/det目录下,如果是/src/abc/只会把abc目录下的文件复制到/det,delete是删除/abc目录下没有而/ldet下有的文件

rsync daemon模式可以在被同步的机子开启rsyncd服务等待客户端连接,然后在客户端安装rsync与inotify-tools来实时监控客户端的数据目录文件增改来同步到rsync服务端

systemd使用的unit分类

centos 7 后,redhad就启用了旧版的system V 服务管理,改用systemd,优点

并行处理所有服务,加速开机服务

旧的init启动脚本是一项一项任务启动,现在都是多核内核架构了，一项一项太慢

一经要求就响应的on-daemon启动方式

systemd全部仅有一个system搭配systemctl命令处理,且systemd常驻内存,效率高

服务依赖性的知觉查检查

B记动依赖A，若A没启动仅手动启动B,system会自动帮启动A

依daemon的分类

依据service,socket,target,path等分类

将多个daemons集合成为一个群组(target)

集合多个daemons，即执行某个target就是执行多个daemon的意思

向下兼容旧的Init服务脚本

注意

运行级别对应上,大概仅有1,3,5有对应system的某些target类弄(与旧片相比)

全部的system都由systemctl这个管理程序管理,systemctl支持语法有限,不像旧的/etc/init.d/daemon可自定参数,systemd不能自定参数

如果某个服务手动去执行如(手动输入crond启动crond服务),那systemd无法检测该服务就无法进一步管理

systemd启动过程中,无法与管理员通过标准输入信息,所以自行偏写systemd启动设置时,要取消交互机制

systemd的配置文件放置目录

/usr/lib/systemd/system 每个服务的最主要的启动脚本设置,类似以前的/etc/init.d下面的文件

/run/systemd/system 系统执行过程中的产生的服和脚本,优先级比上面的高

/etc/systemd/system 管理员概据主机的系统的需求建立的服务脚本，执行优先级比/run/systemd/system高,系统启不启动某些服务其实是看这里,里面是一堆链接文件,而实际执行的systmed启动脚本配置文件其实是放在/usr/lib/systemd/system,所以要修改是去/usr/lib/systemd/system修改,/etc/systemd/system仅是链接到正确的执行脚本配置文件而已

systemd unit类型分类说明

.service

一般服务类型,包括服务器本身的本地服务用网络服务,也是最常见的类型

.socket

内部数据交换的socket服务

.target

执行环境类型,其实是一群unit的集合如multi-user.tartget,执行它就是执行一堆服务

.mount

.automount

.path

检测特定文件或目灵的类型

.timer

循环执行的服务

通过systemctl管理服务

管理单一服务的启动/开机启动与查看状态

[root@study ~]# systemctl [command] [unit]

command主要有：

start ：立刻启动后面接的unit

stop ：立刻关闭后面接的unit

restart ：立刻关闭后启动后面接的unit，亦即执行stop 再start 的意思

reload ：不关闭后面接的unit 的情况下，重新载入设定档，让设定生效

enable ：设定下次开机时，后面接的unit 会被启动

disable ：设定下次开机时，后面接的unit 不会被启动

status ：目前后面接的这个unit 的状态，会列出有没有正在执行、开机预设执行否、登录等资讯等！

is-active ：目前有没有正在运作中

is-enabled：开机时有没有预设要启用这个unit

范例一：看看目前atd这个服务的状态为何？

[root@study ~]# systemctl status atd.service

atd.service - Job spooling tools

Loaded: loaded (/usr/lib/systemd/system/atd.service; enabled)

Active: active (running) since Mon 2015-08-10 19:17:09 CST; 5h 42min ago

Main PID: 1350 (atd)

CGroup: /system.slice/atd.service

└─1350 /usr/sbin/atd -f

Aug 10 19:17:09 study.centos.vbird systemd[1]: Started Job spooling tools.

# 重点在第二、三行喔～

# Loaded：这行在说明，开机的时候这个unit 会不会启动，enabled 为开机启动，disabled 开机不会启动

# Active：现在这个unit 的状态是正在执行(running) 或没有执行(dead)

# 后面几行则是说明这个unit 程序的PID 状态以及最后一行显示这个服务的登录档资讯！

# 登录档资讯格式为：『时间』 『讯息发送主机』 『哪一个服务的讯息』 『实际讯息内容』

# 所以上面的显示讯息是：这个atd 预设开机就启动，而且现在正在运作的意思！

范例二：正常关闭这个atd服务

[root@study ~]# systemctl stop atd.service

[root@study ~]# systemctl status atd.service

atd.service - Job spooling tools

Loaded: loaded (/usr/lib/systemd/system/atd.service; enabled )

Active: inactive (dead) since Tue 2015-08-11 01:04:55 CST; 4s ago

Process: 1350 ExecStart=/usr/sbin/atd -f $OPTS (code=exited, status=0/SUCCESS)

Main PID: 1350 (code=exited, status=0/SUCCESS)

Aug 10 19:17:09 study.centos.vbird systemd[1]: Started Job spooling tools.

Aug 11 01:04:55 study.centos.vbird systemd[1]: Stopping Job spooling tools...

Aug 11 01:04:55 study.centos.vbird systemd[1]: Stopped Job spooling tools.

#目前这个unit下次开机还是会启动，但是现在是没在运作的状态中！同时，

# 最后两行为新增加的登录讯息，告诉我们目前的系统状态喔！

除了running 跟dead 之外， 有没有其他的状态呢？有的～基本上有几个常见的状态：

active (running)：正有一只或多只程序正在系统中执行的意思，举例来说，正在执行中的vsftpd就是这种模式。

active (exited)：仅执行一次就正常结束的服务，目前并没有任何程序在系统中执行。举例来说，开机或者是挂载时才会进行一次的quotaon功能，就是这种模式！quotaon不须一直执行～只须执行一次之后，就交给档案系统去自行处理啰！通常用bash shell写的小型服务，大多是属于这种类型(无须常驻记忆体)。

active (waiting)：正在执行当中，不过还再等待其他的事件才能继续处理。举例来说，列印的伫列相关服务就是这种状态！虽然正在启动中，不过，也需要真的有伫列进来(列印工作)这样他才会继续唤醒印表机服务来进行下一步列印的功能。

inactive：这个服务目前没有运作的意思。

既然daemon 目前的状态就有这么多种了，那么daemon 的预设状态有没有可能除了enable/disable 之外，还有其他的情况呢？当然有！

enabled：这个daemon将在开机时被执行

disabled：这个daemon在开机时不会被执行

static：这个daemon不可以自己启动(enable不可)，不过可能会被其他的enabled的服务来唤醒(相依属性的服务)

mask：这个daemon无论如何都无法被启动！因为已经被强制注销(非删除)。可透过systemctl unmask方式改回原本状态

通过mask功能,可以不必管其他的依赖服务,虽然是非正规,但很有效

# 1.先看看cups的服务是开还是关？

[root@study ~]# systemctl status cups.service

cups.service - CUPS Printing Service

Loaded: loaded (/usr/lib/systemd/system/cups.service; enabled)

Active: inactive (dead) since Tue 2015-08-11 19:19:20 CST; 3h 29min ago

# 有趣得很！竟然是enable 但是却是inactive 耶！相当特别！

# 2.那就直接关闭，同时确认没有启动喔！

[root@study ~]# systemctl stop cups.service

[root@study ~]# systemctl disable cups.service

rm '/etc/systemd/system/multi-user.target.wants/cups.path'

rm '/etc/systemd/system/sockets.target.wants/cups.socket'

rm '/etc/systemd/system/printer.target.wants/cups.service'

# 也是非常特别！竟然一口气取消掉三个连结档！也就是说，这三个档案可能是有相依性的问题喔！

[root@study ~]# netstat -tlunp | grep cups

#现在应该不会出现任何资料！因为根本没有cups的任务在执行当中～所以不会有port产生

# 3.尝试启动cups.socket监听用户端的需求喔！

[root@study ~]# systemctl start cups.socket

[root@study ~]# systemctl status cups.service cups.socket cups.path

cups.service - CUPS Printing Service

Loaded: loaded (/usr/lib/systemd/system/cups.service; disabled)

Active: inactive (dead) since Tue 2015-08-11 22:57:50 CST; 3min 41s ago

cups.socket - CUPS Printing Service Sockets

Loaded: loaded (/usr/lib/systemd/system/cups.socket; disabled)

Active: active (listening) since Tue 2015-08-11 22:56:14 CST; 5min ago

cups.path - CUPS Printer Service Spool

Loaded: loaded (/usr/lib/systemd/system/cups.path; disabled)

Active: inactive (dead)

#确定仅有cups.socket在启动，其他的并没有启动的状态！

# 4.尝试使用lp这个指令来列印看看？

[root@study ~]# echo "testing" | lp

lp: Error - no default destination available. #实际上就是没有印表机！所以有错误也没关系！

[root@study ~]# systemctl status cups.service

cups.service - CUPS Printing Service

Loaded: loaded (/usr/lib/systemd/system/cups.service; disabled)

Active: active (running) since Tue 2015-08-11 23:03:18 CST; 34s ago

[root@study ~]# netstat -tlunp | grep cups

tcp 0 0 127.0.0.1:631 0.0.0.0:\* LISTEN 25881/cupsd

tcp6 0 0 ::1:631 :::\* LISTEN 25881/cupsd

# 见鬼！竟然cups 自动被启动了！明明我们都没有驱动他啊！怎么回事啊？

上面这个范例的练习在让您了解一下，很多服务彼此之间是有相依性的！cups 是一种列印服务，这个列印服务会启用port 631 来提供网路印表机的列印功能。但是其实我们无须一直启动631 埠口吧？因此，多了一个名为cups.socket 的服务，这个服务可以在『用户有需要列印时，才会主动唤醒cups.service 』的意思！因此，如果你仅是disable/stop cups.service 而忘记了其他两个服务的话，那么当有用户向其他两个cups.path, cups.socket 提出要求时， cups.service 就会被唤醒！

通过systemctl查看系统上所有的服务

[root@study ~]# systemctl [command] [--type=TYPE] [--all]

command:

list-units ：依据unit 列出目前有启动的unit。若加上--all 才会列出没启动的。

list-unit-files ：依据/usr/lib/systemd/system/ 内的档案，将所有档案列表说明。

--type=TYPE：就是之前提到的unit type，主要有service, socket, target 等

systemctl list-units --type=service --all | grep iptables #--type字段不要也行

iptables.service loaded active exited IPv4 firewall with iptables

#loaded 开机启动 active正在运行

通过systemctl管理不同的操作环境(target unit)

systemctl list-units --type=target --all

[root@study ~]# systemctl [command] [unit.target]

选项与参数：

command:

get-default ：取得目前的target

set-default ：设定后面接的target 成为预设的操作模式

isolate ：切换到后面接的模式

范例一：我们的测试机器预设是图形界面，先观察是否真为图形模式，再将预设模式转为文字界面

[root@study ~]# systemctl get-default

graphical.target #果然是图形界面喔！

[root@study ~]# systemctl set-default multi-user.target

[root@study ~]# systemctl get-default

multi-user.target

范例二：在不重新开机的情况下，将目前的操作环境改为纯文字模式，关掉图形界面

[root@study ~]# systemctl isolate multi-user.target

范例三：若需要重新取得图形界面呢？

[root@study ~]# systemctl isolate graphical.target

通过systemctl分析各服务之间的依赖性

[root@study ~]# systemctl list-dependencies [unit] [--reverse]

选项与参数：

--reverse ：反向追踪谁使用这个unit 的意思！

范例一：列出目前的target环境下，用到什么特别的unit

[root@study ~]# systemctl get-default

multi-user.target

[root@study ~]# systemctl list-dependencies

default.target

├─abrt-ccpp.service

├─abrt-oops.service

├─vsftpd.service

......

与systemd的daemon运行过程相关的目录

/usr/lib/systemd/system 默认安装的软件的启动脚本配置文件放这里,这里的数据尽量不要改,要修就到/etc/systemd/system下面修改比较好

/run/systemd/system 系统执行过程中所产生的服务脚本,优先级比上面要高

/etc/systemd/system 管理员依据主机系统的需求所建立的执行脚本,优先级比/run/systemd/system高

/etc/sysconf/\* 几乎所有的服务都会将初始化的一些选项设置写入到这里,如网络的数据

/var/lib/ 一些会产生数据的服务都会将它的数据写入到/var/lib目录中,如数拓库默认就写这里

/run 放置了好多的daemon的缓存,包括lock文件以及PID文件等

查看socket文件放哪

[root@localhost 1]# systemctl list-sockets

LISTEN UNIT ACTIVATES

/dev/log systemd-journald.socket systemd-journald.service

/run/dbus/system\_bus\_socket dbus.socket dbus.service

/run/dmeventd-client dm-event.socket dm-event.service

/run/dmeventd-server dm-event.socket dm-event.service

/run/lvm/lvmetad.socket lvm2-lvmetad.socket lvm2-lvmetad.service

......

#这样就清楚在监听本地服务需求的socket文件在哪里

cat /etc/services #查看服务与端口配置文件,建议不要修改,可能会出错

关闭网络服务

[root@study ~]# netstat -tlunp

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN 1340/sshd

tcp 0 0 127.0.0.1:25 0.0.0.0:\* LISTEN 2387/master

tcp6 0 0 :::555 :::\* LISTEN 29113/vsftpd

tcp6 0 0 :::22 :::\* LISTEN 1340/sshd

tcp6 0 0 ::1:25 :::\* LISTEN 2387/master

udp 0 0 0.0.0.0:5353 0.0.0.0:\* 750/avahi-daemon: r

udp 0 0 0.0.0.0:36540 0.0.0.0:\* 750/avahi-daemon: r

[root@study ~]# systemctl list-units --all | grep avahi-daemon

avahi-daemon.service loaded active running Avahi mDNS/DNS-SD Stack

avahi-daemon.socket loaded active running Avahi mDNS/DNS-SD Stack Activation Socket

[root@study ~]# systemctl stop avahi-daemon.service

[root@study ~]# systemctl stop avahi-daemon.socket

[root@study ~]# systemctl disable avahi-daemon.service avahi-daemon.socket

[root@ study ~]# netstat -tlunp

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN 1340/sshd

tcp 0 0 127.0.0.1:25 0.0.0.0:\* LISTEN 2387/master

tcp6 0 0 :::555 :::\* LISTEN 29113/vsftpd

tcp6 0 0 :::22 :::\* LISTEN 1340/sshd

tcp6 0 0 ::1:25 :::\* LISTEN 2387/master

systemctl针对service类型的配置文件

配置文件相关目录

建议修改/etc/systemd/system目录就好

其它参考鸟哥linux书

两个vsftpd 运作的实例

# 1.先建立好所需要的设定档

[root@study ~]# cd /etc/vsftpd

[root@study vsftpd]# cp vsftpd.conf vsftpd2.conf

[root@study vsftpd]# vim vsftpd.conf

#listen\_port =555

[root@study vsftpd]# diff vsftpd.conf vsftpd2.conf

128c128

< #listen\_port=555

---

> listen\_port=555

# 注意这两个设定档的差别喔！只有这一行不同而已！

# 2.开始处理启动脚本设定

[root@study vsftpd]# cd /etc/systemd/system

[root@study system]# cp /usr/lib/systemd/system/vsftpd.service vsftpd2.service

[root@study system ]# vim vsftpd2.service

[Unit]

Description=Vsftpd second ftp daemon

After=network.target

[Service]

Type=forking

ExecStart=/usr/sbin/vsftpd /etc/vsftpd/vsftpd2.conf

[Install]

WantedBy=multi-user.target

# 重点在改了vsftpd2.conf 这个设定档喔！

# 3.重新载入systemd的脚本设定档内容

[root@study system]# systemctl daemon-reload

[root@study system]# systemctl list-unit-files --all | grep vsftpd

vsftpd.service enabled

vsftpd2.service disabled

vsftpd@.service disabled

vsftpd.target disabled

[root@study system]# systemctl status vsftpd2.service

vsftpd2.service - Vsftpd second ftp daemon

Loaded: loaded (/etc/systemd/system/vsftpd2.service; disabled)

Active: inactive (dead)

[root@study system]# systemctl restart vsftpd.service vsftpd2.service

[root@study system]# systemctl enable vsftpd.service vsftpd2.service

[root@study system]# systemctl status vsftpd.service vsftpd2.service

vsftpd.service - Vsftpd ftp daemon

Loaded: loaded (/usr/lib/systemd/system/vsftpd.service; enabled)

Active: active (running) since Wed 2015-08-12 22:00:17 CST; 35s ago

Main PID: 12670 (vsftpd)

CGroup: /system.slice/vsftpd.service

└─12670 /usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf

Aug 12 22:00:17 study.centos.vbird systemd[1]: Started Vsftpd ftp daemon.

vsftpd2.service - Vsftpd second ftp daemon

Loaded: loaded (/etc/systemd/system/vsftpd2.service; enabled)

Active: active (running) since Wed 2015-08-12 22:00:17 CST; 35s ago

Main PID: 12672 (vsftpd)

CGroup: /system.slice/vsftpd2.service

└─12672 /usr/sbin/vsftpd /etc/vsftpd/vsftpd2.conf

[root@study system]# netstat -tlnp

Active Internet connections (only servers)

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN 1340/sshd

tcp 0 0 127.0.0.1:25 0.0.0.0:\* LISTEN 2387/master

tcp6 0 0 :::555 :::\* LISTEN 12672/vsftpd

tcp6 0 0 :::21 :::\* LISTEN 12670/vsftpd

tcp6 0 0 :::22 :::\* LISTEN 1340/sshd

tcp6 0 0 ::1:25 :::\* LISTEN 2387/master

多重的重复设置方式

/usr/lib/systemd/system/getty@.service #详细参考鸟哥

自做服务

[root@study ~]# vim /backups/backup.sh

#!/bin/bash

source="/etc /home /root /var/lib /var/spool/{cron,at,mail}"

target="/backups/backup-system-$(date +%Y-%m-%d).tar.gz"

[ ! -d /backups ] && mkdir /backups

tar -zcvf ${target} ${source} &> /backups/backup.log

[root@study ~]# chmod a+x /backups/backup.sh

[root@study ~]# vim /etc/systemd/system/backup.service

[Unit]

Description=backup my server

Requires=atd.service

[Service]

Type=simple

ExecStart=/bin/bash -c " echo /backups/backup.sh | at now"

[Install]

WantedBy=multi-user.target

#因为ExecStart里面有用到at这个指令，因此， atd.service就是一定要的服务！

[root@study ~]# systemctl daemon-reload

[root@study ~]# systemctl start backup.service

[root@study ~]# systemctl status backup.service

backup.service - backup my server

Loaded: loaded (/etc/systemd/system/backup.service; disabled)

Active: inactive (dead)

Aug 13 07:50:31 study.centos.vbird systemd[1]: Starting backup my server...

Aug 13 07:50:31 study.centos.vbird bash[20490]: job 8 at Thu Aug 13 07:50:00 2015

Aug 13 07:50:31 study.centos.vbird systemd[1]: Started backup my server.

# 为什么Active 是inactive 呢？这是因为我们的服务仅是一个简单的script 啊！

# 因此执行完毕就完毕了，不会继续存在记忆体中喔！

以后你都可以直接使用systemctl start backup.service 进行系统的备份了！而且会直接丢进atd 的管理中， 你就无须自己手动用at 去处理这项任务了

systemctl针对timer的配置

times.target也可以周期性执行任务和crond一样,可以精确毫秒,略过

认识与分析日志文件

记录系统活动信息的几个文件,如何时何地(来源IP),何人(什么服务名称),做了什么操作(信息录录)

重要性

解决系统文面的错误

解决网络服务的问题

过往事件记录

常见的日志文件

/var/log/boot.log

开机启动时候系统内核会去检测与启动硬件,然后记动各种内核功能等,这些流程都会记录到这里

/var/log/cron

/var/log/dmesg

记录系统在开机的时候内核检测过程中所产生的各项信息

/var/log/lastlog

记录系统上面所有的账号最近一次登录的相关信息

/var/log/maillog或/var/log/mail/\*

/var/log/message

相当重要,几乎系统发生的错误信息(重要信息)都记在这里

/var/log/secure

只要牵涉到(需要输入账号密码)的软件,那么当登录时(不管正确与否)都会记录到这里

/var/log/wtmp /var/log/failog

记录正确侠录的信息与错误登录的信息

针对日志文件所需的服务与程序

systemd-journald 最主要的信息记录者,由systemd提供

rsyslog.service 主要收集登录系统与网络的信息

logrotate 主要进行日志文件的轮循功能

centos 7.x 除了即有的rsyslog外,其实最上层还使用了systemd自己的日志文件管理功能,使用的是systmed-jourald这个用务,系统由systemd所管理,所有经由systemd启动的服务,如再启动或结束的过程中发生一些问题或是正常的信息,就会将该信息邮systemd-jourald.service以二进制的方式记录下来,之后再将这个信息发送给rsyslog.service

systemd-journald的记录主要都放置内存中,性能效好

日志文件内容的一般格式

事件发生的日期与时间

发生此事件的主机名

因为日志文件可以做成日志文件服务器,可以收集来自其他服务器的日志文件数据,用来区别信息来源主机

启动此事件的服务名称(如systemd,crond等),或命令与函数名称(如su,login)

该信息的实际内容

rsyslog.service 记录日志文件的服务

[root@study ~]# ps aux | grep rsyslog

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

root 750 0.0 0.1 208012 4732 ? Ssl Aug17 0:00 /usr/sbin/rsyslogd -n

#瞧！确实有启动的！daemon执行档名为rsyslogd喔！

[root@study ~]# systemctl status rsyslog.service

rsyslog.service - System Logging Service

Loaded: loaded (/usr/lib/systemd/system/rsyslog.service; enabled )

Active: active (running) since Mon 2015-08-17 18:37:58 CST; 2 days ago

Main PID: 750 (rsyslogd)

CGroup: /system.slice/rsyslog.service

└─750 /usr/sbin/rsyslogd -n

# 也有启动这个服务，也有预设开机时也要启动这个服务！OK！正常没问题！！

rsyslog.service的配置文件/etc/rsyslog.conf

服务名称.[=!]信息等级 信息记录的档名或装置或主机

#底下以mail这个服务产生的info等级为例：

mail.info /var/log/maillog\_info

# 这一行说明：mail 服务产生的大于等于info 等级的讯息，都记录到

# /var/log/maillog\_info 档案中的意思。

服务名称

相对序号 服务类别 说明

0 kern(kernel) 就是核心(kernel) 产生的讯息，大部分都是硬体侦测以及核心功能的启用

1 user 在使用者层级所产生的资讯，例如后续会介绍到的用户使用logger 指令来记录登录档的功能

2 mail 只要与邮件收发有关的讯息记录都属于这个；

3 daemon 主要是系统的服务所产生的资讯，例如systemd 就是这个有关的讯息！

4 auth 主要与认证/授权有关的机制，例如login, ssh, su 等需要帐号/密码的咚咚；

5 syslog 就是由syslog 相关协定产生的资讯，其实就是rsyslogd 这支程式本身产生的资讯啊！

6 lpr 亦即是列印相关的讯息啊！

7 news 与新闻群组伺服器有关的东西；

8 uucp 全名为Unix to Unix Copy Protocol，早期用于unix 系统间的程序资料交换；

9 cron 就是例行性工作排程cron/at 等产生讯息记录的地方；

10 authpriv 与auth 类似，但记录较多帐号私人的资讯，包括pam 模组的运作等！

11 ftp 与FTP 通讯协定有关的讯息输出！

16~23 local0 ~ local7 保留给本机用户使用的一些登录档讯息，较常与终端机互动。

上面谈到的都是Linux 核心的syslog 函数自行制订的服务名称，软体开发商可以透过呼叫上述的服务名称来记录他们的软体。举例来说， sendmail 与postfix 及dovecot 都是与邮件有关的软体，这些软体在设计登录档记录时，都会主动呼叫syslog 内的mail 服务名称(LOG\_MAIL)。所以上述三个软体(sendmail, postfix, dovecot) 产生的讯息在syslog 看起来，就会『是mail 』类型的服务了

信息等级

等级数值 等级名称 说明

7 debug 用来debug (除错) 时产生的讯息资料；

6 info 仅是一些基本的讯息说明而已；

5 notice 虽然是正常资讯，但比info 还需要被注意到的一些资讯内容；

4 warning

(warn) 警示的讯息，可能有问题，但是还不至于影响到某个daemon 运作的资讯；基本上， info, notice, warn 这三个讯息都是在告知一些基本资讯而已，应该还不至于造成一些系统运作困扰；

3 err

(error) 一些重大的错误讯息，例如设定档的某些设定值造成该服务服法启动的资讯说明， 通常藉由err 的错误告知，应该可以了解到该服务无法启动的问题呢！

2 crit 比error 还要严重的错误资讯，这个crit 是临界点(critical) 的缩写，这个错误已经很严重了喔！

1 alert 警告警告，已经很有问题的等级，比crit 还要严重！

0 emerg

(panic) 疼痛等级，意指系统已经几乎要当机的状态！很严重的错误资讯了。通常大概只有硬体出问题，导致整个核心无法顺利运作，就会出现这样的等级的讯息吧！

基本上，在0(emerg)到6(info)的等级之间，等级数值越高代表越没事，等级靠近0则代表事情大条了！除了0到6之外还有两个比较特殊的等级，那就是debug(错误侦测等级)与none (不需登录等级)两个，当我们想要作一些错误侦测，或者是忽略掉某些服务的资讯时，就用这两个咚咚吧！

.[=!] 的连结符号喔！他代表的意思是这样的：

.：代表『比后面还要严重的等级(含该等级)都被记录下来』的意思，例如： mail.info代表只要是mail的资讯，而且该资讯等级严重于info (含info本身)时，就会被记录下来的意思。

.=：代表所需要的等级就是后面接的等级而已，其他的不要！

.!：有点反向选择的感觉，代表忽略大于等于这个等级的讯息！亦即是低于这个等级的才会被纪录的意思！

一般来说，我们比较常使用的是『.』这个连结符号啦！^\_^

信息记录的文件名或设备名称

档案的绝对路径：通常就是放在/var/log 里头的档案啦！

印表机或其他：例如/dev/lp0 这个印表机装置

使用者名称：显示给使用者啰！

远端主机：例如@study.vbird.tsai 当然啦，要对方主机也能支援才行！

\*：代表『目前在线上的所有人』，类似wall 这个指令的意义！

服务 daemon 名称

syslog 这个是Linux 核心所提供的登录档设计指引，所有的要求大概都写入道一个名为syslog.h 的标头档案中。如果你想要开发与登录档有关的软体， 那你就得要依循这个syslog 函数的要求去设计才行！可以使用man 3 syslog 去查询一下相关的资料！

rsyslogd 为了要达成实际上进行讯息的分类所开发的一套软体，所以，这就是最基本的daemon 服务

rsyslog.service 为了加入systemd 的控制，因此rsyslogd 的开发者设计的启动服务脚本配置文件

例题：

如果我要将我的mail 相关的资料给他写入/var/log/maillog 当中，那么在/etc/rsyslog.conf 的语法如何设计？

答：

基本的写法是这样的：

mail.info /var/log/maillog

例题：

我要将新闻群组资料(news) 及例行性工作排程(cron) 的讯息都写入到一个称为/var/log/cronnews 的档案中，但是这两个程序的警告讯息则额外的记录在/var/log/cronnews.warn 中， 那该如何设定我的rsyslog.conf 呢？

答：

很简单啦！既然是两个程序，那么只好以分号来隔开了，此外，由于第二个指定档案中，我只要记录警告讯息， 因此设定上需要指定『.=』这个符号，所以语法成为了：

news.\*;cron.\* /var/log/cronnews

news.=warn;cron.=warn /var/log/cronnews.warn

例题：

我的messages 这个档案需要记录所有的资讯，但是就是不想要记录cron, mail 及news 的资讯，那么应该怎么写才好？

答：

可以有两种写法，分别是：

\*.\*;news,cron,mail.none /var/log/messages

\*.\*;news.none;cron.none;mail.none /var/log/messages

使用『,』分隔时，那么等级只要接在最后一个即可，如果是以『;』来分的话， 那么就需要将服务与等级都写上去啰！这样会设定了吧！

centos 7 默认的rsysog.conf内容

[root@study ~]# vim /etc/rsyslog.conf

1 #kern.\* /dev/console

2 \*.info;mail.none;authpriv.none;cron.none /var /log/messages

3 authpriv.\* /var/log/secure

4 mail.\* -/var/log/maillog

5 cron.\* /var/log/cron

6 \*.emerg :omusrmsg:\*

7 uucp,news.crit / var/log/spooler

8 local7.\* /var/log/boot.log

上面总共仅有8 行设定值，每一行的意义是这样的：

#kern.\*：只要是核心产生的讯息，全部都送到console(终端机)去。console通常是由外部装置连接到系统而来，举例来说，很多封闭型主机(没有键盘、萤幕的系统)可以透过连接RS232连接口将讯息传输到外部的系统中，例如以笔记型电脑连接到封闭主机的RS232插口。这个项目通常应该是用在系统出现严重问题而无法使用预设的萤幕观察系统时，可以透过这个项目来连接取得核心的讯息。( 注1 )

\*.info;mail.none;authpriv.none;cron.none：由于mail, authpriv, cron 等类别产生的讯息较多， 且已经写入底下的数个档案中，因此在/var/log/messages 里面就不记录这些项目。除此之外的其他讯息都写入/var/log/messages 中。这也是为啥我们说这个messages 档案很重要的缘故！

authpriv.\*：认证方面的讯息均写入/var/log/secure 档案；

mail.\*：邮件方面的讯息则均写入/var/log/maillog 档案；

cron.\*：例行性工作排程均写入/var/log/cron 档案；

\*.emerg：当产生最严重的错误等级时，将该等级的讯息以wall 的方式广播给所有在系统登入的帐号得知， 要这么做的原因是希望在线的使用者能够赶紧通知系统管理员来处理这么可怕的错误问题。

uucp,news.crit：uucp 是早期Unix-like 系统进行资料传递的通讯协定，后来常用在新闻群组的用途中。news 则是新闻群组。当新闻群组方面的资讯有严重错误时就写入/var/log/spooler 档案中；

local7.\*：将本机开机时应该显示到萤幕的讯息写入到/var/log/boot.log 档案中；

在上面的第四行关于mail的记录中，在记录的档案/var/log/maillog前面还有个减号『 - 』是干嘛用的？由于邮件所产生的讯息比较多，因此我们希望邮件产生的讯息先储存在速度较快的记忆体中(buffer) ，等到资料量够大了才一次性的将所有资料都填入磁碟内，这样将有助于登录档的存取性能。只不过由于讯息是暂存在记忆体内，因此若不正常关机导致登录资讯未回填到登录档中，可能会造成部分资料的遗失。

自行增加日志

让所有的信息都额外写入到/var/log/admin.log

# 1.先设定好所要建立的档案设置！

[root@study ~]# vim /etc/rsyslog.conf

# Add by VBird 2015/08/19 <==再次强调，自己修改的时候加入一些说明

\*.info /var/log/admin.log <==有用的是这行啦！

# 2.重新启动rsyslogd呢！

[root@study ~]# systemctl restart rsyslog.service

[root@study ~]# ll /var/log/admin.log

-rw-r--r--. 1 root root 325 Aug 20 00:54 /var/log/admin.log

# 瞧吧！建立了这个登录档出现啰！

写进rsyslog.conf文件即可

日志文件安全性设置

一般默客入侵系统后,会清除相关日志来(清除证据),所以我们可以让日志只能增加不能修改与删除

chattr +a /var/log/admin.log

这样也可以避免被root不小心以vi方式打开修改日志文件,以vi修改日志文件后,rsyslog会误判文件被修改过就不写数据进去了

虽然chattr +a 可以加强保护,但是如果root账密被黑客登录,chattr也没用,因为root有所有权限,而且chattr +a 后，日志文件不能修改与删除,日志轮循时也将无法重命名,不过logrotate轮循可以配置实现相关

日志文件服务器的设置

如有十台提供服务的Linux主机,可以拿一台作为日志文件服务器的server来接收其它主机的日志,这样就不用一台一台去查看日志了

# 1. Server端：修改rsyslogd的启动设定档，在/etc/rsyslog.conf内！

[root@study ~]# vim /etc/rsyslog.conf

#找到底下这几行：

# Provides UDP syslog reception

#$ModLoad imudp

#$UDPServerRun 514

# Provides TCP syslog reception

#$ModLoad imtcp

#$InputTCPServerRun 514

# 上面的是UDP 埠口，底下的是TCP 埠口！如果你的网路状态很稳定，就用UDP 即可。

#不过，如果你想要让资料比较稳定传输，那么建议使用TCP啰！所以修改底下两行即可！

$ModLoad imtcp

$InputTCPServerRun 514

# 2.重新启动与观察rsyslogd喔！

[root@study ~]# systemctl restart rsyslog.service

[root@study ~]# netstat -ltnp | grep syslog

Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name

tcp 0 0 0.0.0.0: 514 0.0.0.0:\* LISTEN 2145/rsyslogd

tcp6 0 0 :::514 :::\* LISTEN 2145/rsyslogd

# 嘿嘿！你的登录档主机已经设定妥当啰！很简单吧！

假设server IP 为192.168.1.100

Client 设置

[root@study ~]# vim /etc/rsyslog.conf

\*.\* @@192.168.1.100

#\*.\* @192.168.1.100 #若用UDP传输，设定要变这样！

[root@study ~]# systemctl restart rsyslog.service

日志文件的轮询(logrotate)

配置文件

/etc/logrotate.conf

/etc/logrotate.d/

[root@study ~]# vim /etc/logrotate.conf

#底下的设定是"logrotate的预设设定值" ，如果个别的档案设定了其他的参数，

# 则将以个别的档案设定为主，若该档案没有设定到的参数则以这个档案的内容为预设值！

weekly <==预设每个礼拜对登录档进行一次rotate的工作

rotate 4 <==保留几个登录档呢？预设是保留四个！

create <==由于登录档被更名，因此建立一个新的来继续储存之意！

dateext <==就是这个设定值！可以让被轮替的档案名称加上日期作为档名喔！

#compress <==被更动的登录档是否需要压缩？如果登录档太大则可考虑此参数启动

include /etc/logrotate.d

# 将/etc/logrotate.d/ 这个目录中的所有档案都读进来执行rotate 的工作！

/var/log/wtmp { <==仅针对/var/log/wtmp所设定的参数

monthly <==每个月一次，取代每周！

create 0664 root utmp <==指定新建档案的权限与所属帐号/群组

minsize 1M <==档案容量一定要超过1M后才进行rotate (略过时间参数)

rotate 1 <==仅保留一个，亦即仅有wtmp.1保留而已。

}

#这个wtmp可记录登入者与系统重新开机时的时间与来源主机及登入期间的时间。

# 由于具有minsize 的参数，因此不见得每个月一定会进行一次喔！要看档案容量。

# 由于仅保留一个登录档而已，不满意的话可以将他改成rotate 5 吧！

[root@study ~]# vim /etc/logrotate.d/syslog

/var/log/cron

/var/log/maillog

/var/log/messages

/var/log/secure

/var/log/spooler

{

sharedscripts

postrotate

/bin/kill -HUP `cat /var/run/syslogd.pid 2> /dev/null` 2> /dev/null || true

endscript

}

档名：被处理的登录档绝对路径档名写在前面，可以使用空白字元分隔多个登录档；

参数：上述档名进行轮替的参数使用{ }包括起来；

执行脚本：可呼叫外部指令来进行额外的命令下达，这个设定需与sharedscripts .... endscript设定合用才行。至于可用的环境为：

prerotate：在启动logrotate之前进行的指令，例如修改登录档的属性等动作；

postrotate：在做完logrotate之后启动的指令，例如重新启动(kill -HUP)某个服务！

Prerotate 与postrotate 对于已加上特殊属性的档案处理上面，是相当重要的执行程序！

那么/etc/logrotate.d/syslog 内设定的5 个档案的轮替功能就变成了：

该设定只对/var/log/ 内的cron, maillog, messages, secure, spooler 有效；

登录档轮替每周一次、保留四个、且轮替下来的登录档不进行压缩(未更改预设值)；

轮替完毕后(postrotate) 取得syslog 的PID 后，以kill -HUP 重新启动syslogd

假设我们有针对/var/log/messages 这个档案增加chattr +a 的属性时， 依据logrotate 的工作原理，我们知道，这个/var/log/messages 将会被更名成为/var/log/messages.1 才是。但是由于加上这个+a 的参数啊，所以更名是不可能成功的！那怎么办呢？呵呵！就利用prerotate 与postrotate 来进行登录档轮替前、后所需要作的动作啊！果真如此时，那么你可以这样修改一下这个档案喔！

[root@study ~]# vim /etc/logrotate.d/syslog

/var/log/cron

/var/log/maillog

/var/log/messages

/var/log/secure

/var/log/spooler

{

sharedscripts

prerotate

/usr/bin/chattr -a /var/log/messages

endscript

sharedscripts

postrotate

/bin/kill -HUP `cat /var/run/syslogd.pid 2> /dev/null` 2> /dev/null || true

/usr/bin/chattr +a /var/log/messages

endscript

}

看到否？就是先给他去掉a这个属性，让登录档/var/log/messages可以进行轮替的动作，然后执行了轮替之后，再给他加入这个属性！请特别留意的是，那个/bin/kill -HUP ...的意义，这一行的目的在于将系统的rsyslogd重新以其参数档(rsyslog.conf)的资料读入一次！也可以想成是reload的意思啦

实际测试logrotate的操作

[root@study ~]# logrotate [-vf] logfile

选项与参数：

-v ：启动显示模式，会显示logrotate 运作的过程喔！

-f ：不论是否符合设定档的资料，强制每个登录档都进行rotate 的动作！

范例一：执行一次logrotate看看整个流程为何？

[root@study ~]# logrotate -v /etc/logrotate.conf

reading config file /etc/logrotate.conf <==读取主要设定档

including /etc/logrotate.d <==呼叫外部的设定

reading config file chrony <==就是外部设定啊！

....(中间省略)....

Handling 18 logs <==共有18个登录档被记录

....(中间省略)....

rotating pattern: /var/log/cron

/var/log/maillog

/var/log/messages

/var/log/secure

/var/log/spooler

weekly (52 rotations)

empty log files are not rotated, old logs are removed

considering log /var/log/cron

log does not need rotating

considering log /var/log/maillog

log does not need rotating

considering log /var/log/messages <==开始处理messages

log does not need rotating <==因为时间未到，不需要更动！

....(底下省略)....

范例二：强制进行logrotate的动作

[root@study ~]# logrotate -vf /etc/logrotate.conf

....(前面省略)....

rotating log /var/log/messages, log->rotateCount is 52

dateext suffix '-20150820'

glob pattern '-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'

compressing log with: /bin/gzip

....(底下省略)....

#看到否？整个rotate的动作就是这样一步一步进行的～

[root@study ~]# ll /var/log/messages\*; lsattr /var/log/messages

-rw-------. 1 root root 143 Aug 20 01:45 /var/log/messages

-rw-------. 1 root root 167125 Aug 20 01:40 /var/log/messages-20150820

-----a---------- /var/log/messages <==主动加入a的隐藏属性啰！

自定义日志文件的轮循功能

假设你已经建立了/var/log/admin.log 这个档案， 现在，你想要将该档案加上+a 这个隐藏标签，而且设定底下的相关资讯：

登录档轮替一个月进行一次；

该登录档若大于10MB 时，则主动进行轮替，不需要考虑一个月的期限；

保存五个备份档案；

备份档案需要压缩

# 1.先建立+a这个属性啊！

[root@study ~]# chattr +a /var/log/admin.log

[root@study ~]# lsattr /var/log/admin.log

-----a---------- /var/log/admin.log

[root@study ~]# mv /var/log/admin.log /var/log/admin.log.1

mv: cannot move `/var/log/admin.log' to `/var/log/admin.log.1': Operation not permitted

# 这里确定了加入a 的隐藏属性！所以root 无法移动此登录档！

# 2.开始建立logrotate的设定档，增加一个档案在/etc/logrotate.d内就对了！

[root@study ~]# vim /etc/logrotate.d/admin

# This configuration is from VBird 2015/08/19

/var/log/admin.log {

monthly <==每个月进行一次

size=10M <==档案容量大于10M则开始处置

rotate 5 <==保留五个！

compress <==进行压缩工作！

sharedscripts

prerotate

/usr/bin/chattr -a /var/log/admin.log

endscript

sharedscripts

postrotate

/bin/kill -HUP `cat /var/run/syslogd.pid 2> /dev/null` 2> /dev/null || true

/usr/bin/chattr +a /var/log/admin.log

endscript

}

# 3.测试一下logrotate相关功能的资讯显示：

[root@study ~]# logrotate -v /etc/logrotate.conf

....(前面省略)....

rotating pattern: /var/log/admin.log 10485760 bytes (5 rotations)

empty log files are rotated, old logs are removed

considering log /var/log/admin.log

log does not need rotating

not running prerotate script, since no logs will be rotated

not running postrotate script, since no logs were rotated

....(底下省略)....

# 因为还不足一个月，档案也没有大于10M，所以不需进行轮替！

# 4.测试一下强制logrotate与相关功能的资讯显示：

[root@study ~]# logrotate -vf /etc/logrotate.d/admin

reading config file /etc/logrotate.d/admin

reading config file /etc/logrotate.d/admin

Handling 1 logs

rotating pattern: /var/log/admin.log forced from command line (5 rotations)

empty log files are rotated, old logs are removed

considering log /var/log/admin.log

log needs rotating

rotating log /var/log/admin.log, log->rotateCount is 5

dateext suffix '-20150820'

glob pattern '-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'

renaming /var/log/admin.log.5.gz to /var/log/admin.log.6.gz (rotatecount 5, logstart 1, i 5),

old log /var/log/admin.log.5.gz does not exist

renaming /var/log/admin.log.4.gz to /var/log/admin.log.5.gz (rotatecount 5, logstart 1, i 4),

old log /var/log/admin.log.4.gz does not exist

renaming /var/log/admin.log.3.gz to /var/log/admin.log.4.gz (rotatecount 5, logstart 1, i 3),

old log /var/log/admin.log.3.gz does not exist

renaming /var/log/admin.log.2.gz to /var/log/admin.log.3.gz (rotatecount 5, logstart 1, i 2),

old log /var/log/admin.log.2.gz does not exist

renaming /var/log/admin.log.1.gz to /var/log/admin.log.2.gz (rotatecount 5, logstart 1, i 1),

old log /var/log/admin.log.1.gz does not exist

renaming /var/log/admin.log.0.gz to /var/log/admin.log.1.gz (rotatecount 5, logstart 1, i 0),

old log /var/log/admin.log.0.gz does not exist

log /var/log/admin.log.6.gz doesn't exist -- won't try to dispose of it

running prerotate script

fscreate context set to system\_u:object\_r:var\_log\_t:s0

renaming /var/log/admin.log to /var/log/admin.log.1

running postrotate script

compressing log with: /bin/gzip

[root@study ~]# lsattr /var/log/admin.log\*

-----a---------- /var/log/admin.log

---------------- /var/log/admin.log.1.gz <==有压缩过喔！

systemd-journald.service简介

过于由于rsyslogd必须要启动完成后日志文件才开始记录,所以内核还得要自己产生一个klogd的服务才能将系统在启动过程中的信息记录下来,然后等rsyslog启动后才传送给它处理

现在有了systemd之后,由于它是内核呼醒,然后又是第一执行任务,它可以主动调用systmed-journald来协助记录文件,记录开机启动过程中的所有信息与服务启动信息

由于systemd是使用内存的日志记录方式,因此重新启动过后,开机启动前的日志文件信息就不会记录了,所以还是建议启动rsyslogd来协助记录,也就是说systemd-journald用来管理与查询这次启动后的登录信息,rsyslogd可以用来记录以前及现在的所有数据到磁盘文件中,方便未来查询

journaldctl 查看日志信息,详细查鸟哥Linux

logger 将一些数据存储到日志中的命令,如可以在脚本中设置开始与结束时间,详细查鸟哥

journal也可以保存在磁盘中,直接在/var/log新建joural文件即可,这样一来,/run/log就不会存储了,(run内存的数据存放位置),不建议保存,因为有rsyslog了

logwatch

日志分析工具,可以整体查看系统日志,也可以自己定制

启动流程,模块管理与Loader

操作系统启动流程

加载BIOS的硬件信息与进行自我检测,并根据设置取得第一个可启动的设备

读取并执行第一个启动设备内MBR内的启动引导程序(即grub2,spfdisk等程序)

根据启动引导程序的设置加载Kerner(加载到内存当中解压缩与执行),Kerner会开始检测硬件与加载驱动程序

在硬件驱动成功后,Kerner会主动调用systemd程序,并以default.target流程启动

systemd执行sysinit.target初始化系统及basic.target准备操作系统

systmed启动multi-user.target下的本机与服务器服务

systemd执行multi-user.target下的/etc/rc.d/rc.local文件

systemd执行multi-user.target下的getty.target及登录服务

systemd执行graphical需要的服务

BIOS,启动自我测试与MBR/GPT

启动整个系经首先就得要让系统去加载BIOS,并通过BIOS加载CMOS的信息,并由CMOS内的设置值取得主机的各项硬件配置(如CPU与接口设备的沟通频率,启动设备的查找顺序,硬盘的大小与类型等),然后BIOS还会进行自我检测,然后开始执行硬件检测的初始化,之后再定义可启动的设备顺序,接下来就会开始进行启动设备的数据读取了

BIOS会指定启动设备的好让可以读取磁盘中的操作系统内核文件,由于不同操作系统它的文件系统不相同,因此必须要以一个启动引导程序来处理内核文件的加载(load)的问题,这个启动引导程序就被称为(boot loader),boot loader安将在启动设备的第一个扇区(sector)中,也就是MBR

boot loader(启动引导程序)

loader最主要的功能就是识别操作系统的文件格式并据以加载内核到内存中执行,由于不同操作系统文件格式不一样,因此每种操作系统都有自己的boot loader

多重引导

每个文件系统都会保留一块启动扇区(boot sector)提供操作系统安装boot loader,默认安装

先看boot loader主要功能

提供选项 用户可以选择不同的启动选项,这也是多重引导的重要功能

加载内核文件 直接指向可启动的程序区域名来启动操作系统

转交给其他loader 将启动管理功能转交给其他loader负责

linux安装系统时可以选择将boot loader安装到MBR,如果选择则MBR与boot sector都会保留一份boot loader

windows安装时,默认会主动地将MBR与boot sector都装上一份boot loader

因为boot loader具有控制权转交功能,但windows的boot loader不具有控制权转交功能,因此不能使用windows的Loader来加载linux的loader,所以会强调先安装windows再装linux

所以多系统安装引导是会是这样子

選單一：MBR(grub2) --> kernel file --> booting

選單二：MBR(grub2) --> boot sector(Windows loader) --> Windows kernel --> booting

選單三：MBR(grub2) --> boot sector(grub2) --> kernel file --> booting

MBR使用linux的grub2引导程序,里面有3个选项，1选项是可以直接指向linux内核文件并加载核来启动, 2选项将启动管理权交给windows管理,windows loader会接管启动流程,它就能记动windows了, 3选项则是使用Linux在boot sectoer内的启动引导程序,此时就会跳出另一个grub2选项

加载内核检测硬件与initramfs的功能

当由boot loader的管理而开始读取内核文件后,接下来Linux会将内核文件解压缩到内存,并利用内核的功能,开始测试与驱动各周边设备(存付,网卡CPU等),此时Linux内核会以自己的功能重新检测一次硬件,而不一定会使用BIOS检测到的硬件信息,就是说内核此时才开始接管BIOS后的工作

内核文件位置

/boot 名为vmlinuz

linux内核可以通过动态加载内核模块(类似驱动程序),内核模块位置/lib/modules目录中

由于模块放置在磁盘根目录(/lib必须与/在同一分区),因此启动过程中内核必须挂载根目录,这样才能读取内核模块提供的加载驱动程序功能,根目录此时挂载为只读

linux会将非必要的功能且可编译为模块的内核功能,编译成模块,如USB,SATA,SCSI等磁盘设备的驱动程序都是以模块存在,假设linux安装在SATA磁盘,可以通过BIOS的INT 13取得boot loader与内核文件启动,然后内核会开始接管系统并检测硬件及尝试挂载根目录来以得额外的驱动程序,问题是内核根本不认识SATA磁盘,所以需要加载SATA驱动程序否则无法挂载根目录,但SATA的驱动程序在/lib/modules内,这个问题需要虚拟文件系统来处理

虚拟文件系统

一般使用的文件名为/boot/initrd或/boot/initramfs，这文件也能通过被boot loader解压缩加载到内存且在内存中模拟一个根目录,且提供一个程序来加载启动过程中需要的内核模块(通常是USB,RAID,SATA等与文件系统与磁盘接口的驱动程序),加载后会帮助内核重新(并挂载实际的根目录)调用systemd开始后续的正常动流程

其它的虚拟文件详细信息参考鸟哥

第一个程序systemd及使用default.target进入启动程序分析

内核完整地加载后,主机就应该开始正确运行了,接下来就到系统的第一个程序执行了,systemd

systemd最主要的功能就是准备软件的执行环境,包括系统的主机名,网络设置,语文设置等,所有的操作都通过systemd的默认启动服务的集合即/etc/systemd/system/default.target来规划

systemd的处理流程

取得default.target后,它会链接到/usr/lib/systemd/system下去取得multi-user/target或graphical.target其中的一个,假设使用的是graphical.target,接下来systemd会去找两个地方的设置

/etc/systemd/system/graphical.target.wants/ 用户设置加载的unit

/usr/lib/systemd/system/graphical.target.wants/ 系统默认加载的Unit

然后再由/usr/lib/systemd/system/graphical.target配置发现

[root@study ~]# cat /usr/lib/systemd/system/graphical.target

[Unit]

Description=Graphical Interface

Documentation=man:systemd.special(7)

Requires=multi-user.target

After=multi-user.target

Conflicts=rescue.target

Wants=display-manager.service

AllowIsolate=yes

[Install]

Alias=default.target

这表示graphical.target 必须要完成multi-user.target 之后才能够进行，而进行完graphical.target 之后，还得要启动display-manager.service 才行的意思

然后

[root@study ~]# cat /usr/lib/systemd/system/multi-user.target

[Unit]

Description=Multi-User System

Documentation=man:systemd.special(7)

Requires=basic.target

Conflicts=rescue.service rescue.target

After=basic.target rescue.service rescue.target

AllowIsolate=yes

[Install]

Alias=default.target

#然后看看系统预设要载入的unit有哪些？

[root@study ~]# ls /usr/lib/systemd/system/multi-user.target.wants

brandbot.path plymouth-quit.service systemd-logind.service

dbus.service plymouth-quit-wait.service systemd-user-sessions.service

getty.target systemd-ask-password-wall.path

#使用者自订要载入的unit又有哪些呢？

[root@study ~]# ls /etc/systemd/system/multi-user.target.wants

abrt-ccpp.service crond.service mdmonitor.service sshd.service

abrtd.service hypervkvpd.service ModemManager.service sysstat.service

abrt-oops.service hypervvssd.service NetworkManager.service tuned.service

.....

然后又知道multi-usre.target需要在basic.target运行完毕才能加载上述的许多unit,这大概就是systemd的启动流程

基本上我们CentOS 7.x 的systemd 开机流程大约是这样：

local-fs.target + swap.target：这两个target 主要在挂载本机/etc/fstab 里面所规范的文件系统与相关的内存交换分区。

sysinit.target：这个target 主要在检测硬体，加载所需要的内核模块等动作。

basic.target：载入主要的周边硬件驱动程式与防火墙相关任务

multi-user.target 底下的其它一般系统或网路服务的载入

图形界面相关服务如gdm.service 等其他服务的载入

除了第一步骤local-fs.target, swap.target 是透过/etc/fstab 来进行挂载的行为之外，那其他的target 有做啥动作呢？简单得来说说！

systemd执行sysinit.target初始化系统, basick准备系统

sysinit.target

基本的内核功能,文件系统设置的驱动等在这阶段,了解即行

特殊档案系统装置的挂载：包括dev-hugepages.mount dev-mqueue.mount 等挂载服务，主要在挂载跟巨量记忆体分页使用与讯息伫列的功能。挂载成功后，会在/dev 底下建立/dev/hugepages/, /dev/mqueue/ 等目录；

特殊档案系统的启用：包括磁碟阵列、网路磁碟(iscsi)、LVM 档案系统、档案系统对照服务(multipath) 等等，也会在这里被侦测与使用到！

开机过程的讯息传递与动画执行：使用plymouthd 服务搭配plymouth 指令来传递动画与讯息

日志式登录档的使用：就是systemd-journald 这个服务的启用啊！

载入额外的核心模组：透过/etc/modules-load.d/\*.conf 档案的设定，让核心额外载入管理员所需要的核心模组！

载入额外的核心参数设定：包括/etc/sysctl.conf 以及/etc/sysctl.d/\*.conf 内部设定！

启动系统的乱数产生器：乱数产生器可以帮助系统进行一些密码加密演算的功能

设定终端机(console) 字形

启动动态装置管理员：就是udevd 这个家伙！用在动态对应实际装置存取与装置档名对应的一个服务！相当重要喔！也是在这里启动的！

sysinit.target执行之后就到basic.target

basic.target 了解即行

载入alsa 音效驱动程式：这个alsa 是个音效相关的驱动程式，会让你的系统有音效产生啰；

载入firewalld 防火墙：CentOS 7.x 以后使用firewalld 取代iptables 的防火墙设定，虽然最终都是使用iptables 的架构， 不过在设定上面差很多喔！

载入CPU 的微指令功能；

启动与设定SELinux 的安全本文：如果由disable 的状态改成enable 的状态，或者是管理员设定强制重新设定一次SELinux 的安全本文， 也在这个阶段处理喔！

将目前的开机过程所产生的开机资讯写入到/var/log/dmesg 当中

由/etc/sysconfig/modules/\*.modules 及/etc/rc.modules 载入管理员指定的模组！

载入systemd 支援的timer 功能；

systemctl启动multi-user.target下的服务

加载内核驱动硬件后,经过sysinit.target的初始化流程让系统可以读写之后,加上basic.target让系统成为操系统的基础,之后就是服务器要顺利运行时,需要的各种主机服务及提供服务器功能的网络服务启动,这些服务的启动大多是在multi-user.target操作环境下

你可以到/etc/systemd/system/multi-user.target.wants/ 里头去瞧瞧预设要被启动的服务喔！

也就是说，一般来说服务的启动脚本设定都是放在底下的目录内：

/usr/lib/systemd/system (系统预设的服务启动脚本设定)

/etc/systemd/system (管理员自己开发与设定的脚本设定)

而用户针对主机与网络服务的各项unit若要enable的话,就是将它放到/etc/systemd/system/multi-user.target.wants/ 这个目录底下做个连结

#将vsftpd.service先disable再enable看看输出的资讯为何？

[root@study ~]# systemctl disable vsftpd.service

rm ' /etc/systemd/system/multi-user.target.wants/ vsftpd.service'

[root@study ~]# systemctl enable vsftpd.service

ln -s '/usr/lib/systemd/system/vsftpd.service' ' /etc/systemd/system/multi-user.target.

wants/ vsftpd.service'

开机启动运行服务

# 1.先看一下/etc/rc.d/rc.local的权限，然后检查multi-user.target有没有这个服务

[root@study ~]# ll /etc/rc.d/rc.local

-rw -r--r-- . 1 root root 473 Mar 6 13:48 /etc/rc.d/rc.local

[root@study ~]# systemctl status rc-local.service

rc-local.service - /etc/rc.d/rc.local Compatibility

Loaded: loaded (/usr/lib/systemd/system/rc-local.service; static)

Active: inactive (dead)

[root@study ~]# systemctl list-dependencies multi-user.target | grep rc-local

#明明就有这个服务，但是rc.local不具有可执行(x)的权限，因此这个服务不会被执行

# 2.加入可执行权限后，再看一下rc-local是否可被启用！

[root@study ~]# chmod a+x /etc/rc.d/rc.local; ll /etc/rc.d/rc.local

-rwxr-xr-x . 1 root root 473 Mar 6 13:48 / etc/rc.d/rc.local

[root@study ~]# systemctl daemon-reload

[root@study ~]# systemctl list-dependencies multi-user.target | grep rc-local

├─rc-local.service #这个服务确实被记录到启动的环境下啰！

透过这个chmod a+x /etc/rc.d/rc.local 的步骤，你的许多脚本就可以放在/etc/rc.d/rc.local 这个档案内， 系统在每次开机都会去执行这档案内的指令喔！非常简单吧

tty登录界面

tty登录的界面与服务也是在multi-user.target下面的

graphical.target(图形界面管理程序)

如果default.target是multi-user.target的话,这个步骤不会进行

systemd启动流程到这里就完毕

内核模块与依赖性(先了解)

内核模块的放置处是/lib/modules/$(uname -r)/kernel

arch ：与硬体平台有关的项目，例如CPU 的等级等等；

crypto ：核心所支援的加密的技术，例如md5 或者是des 等等；

drivers ：一些硬体的驱动程式，例如显示卡、网路卡、PCI 相关硬体等等；

fs ：核心所支援的filesystems ，例如vfat, reiserfs, nfs 等等；

lib ：一些函式库；

net ：与网路有关的各项协定资料，还有防火墙模组(net/ipv4/netfilter/\*) 等等；

sound ：与音效有关的各项模组；

/lib/modules/$(uname -r)/modules.dep 记录了内核支持的模块的各项依赖性

范例一：若我做好一个网路卡驱动程式，档名为a.ko，该如何更新核心相依性？

[root@study ~]# cp a.ko /lib/modules/$(uname -r)/kernel/drivers/net

[root@study ~]# depmod

查看内核模块

[root@study ~]# lsmod

Module Size Used by

nf\_conntrack\_ftp 18638 0

nf\_conntrack 105702 1 nf\_conntrack\_ftp

....(中间省略)....

qxl 73766 1

drm\_kms\_helper 98226 1 qxl

ttm 93488 1 qxl

drm 311588 4 qxl,ttm,drm\_kms\_helper # drm还被qxl, ttm..等模组使用

....(底下省略)....

使用lsmod 之后，系统会显示出目前已经存在于核心当中的模组，显示的内容包括有：

模组名称(Module)；

模组的大小(size)；

此模组是否被其他模组所使用(Used by)。

范例一：由上个表格当中，请列出drm这个模组的相关资讯：

[root@study ~]# modinfo drm

filename: /lib/modules/3.10.0-229.el7.x86\_64/kernel/drivers/ gpu/drm/drm.ko

license: GPL and additional rights

description: DRM shared core routines

。。。。。

内核的加载与删除

[root@study ~]# modprobe [-cfr] module\_name

选项与参数：

-c ：列出目前系统所有的模组！(更详细的代号对应表)

-f ：强制载入该模组；

-r ：类似rmmod ，就是移除某个模组啰～

范例一：载入vfat模组

[root@study ~]# modprobe vfat

#很方便吧！不需要知道完整的模组档名，这是因为该完整档名已经记录到

# /lib/modules/`uname -r`/modules.dep当中的缘故啊！如果要移除的话：

[root@study ~]# modprobe -r vfat

Boot loader:Grub2

boot loader的两个stage

MBR是整个硬盘的第一个扇区的一个区块,大小也就446B而已,即使GPT也没有很大的扇区来存储loader数据,那如何解决?

linux将boot loader的程序代码执行与设置值加载分成两个阶段

stage 1 执行boot loader主程序

主程序必须安装在启动区,即MBR或启动扇区,但MBR太小,所以仅安装boot loader的最小程序,并没有安装loader相关配置文件

stage 2 主程序加载配置文件

通过boot loader加载所有配置文件与相关的环境参数文件(包括文件系统定义与主要配置文件grub.cfg),一般配置文件都放在/boot下面(boot/grub2)

[root@study ~]# ls -l /boot/grub2

-rw-r--r--. device.map <==grub2的装置对应档(底下会谈到)

drwxr-xr-x. fonts <==开机过程中的画面会使用到的字型资料

-rw-r--r--. grub.cfg <==grub2的主设定档！相当重要！

-rw-r--r--. grubenv <==一些环境区块的符号

drwxr-xr-x. i386-pc <==针对一般x86 PC所需要的grub2的相关模组

drwxr-xr-x. locale <==就是语系相关的资料啰

drwxr-xr-x. themes <==一些开机主题画面资料

[root@study ~]# ls -l /boot/grub2/i386-pc

-rw-r--r--. acpi.mod <==电源管理有关的模组

-rw-r--r--. ata.mod <==磁碟有关的模组

-rw-r--r--. chain.mod <==进行loader控制权移交的相关模组

-rw-r--r--. command.lst <==一些指令相关性的列表

-rw-r--r--. efiemu32.o <==底下几个则是与uefi BIOS相关的模组

-rw-r--r--. efiemu64.o

-rw-r--r--. efiemu.mod

-rw-r--r--. ext2.mod <==EXT档案系统家族相关模组

-rw-r--r--. fat.mod <==FAT档案系统模组

-rw-r-- r--. gcry\_sha256.mod <==常见的加密模组

-rw-r--r--. gcry\_sha512.mod

-rw-r--r--. iso9660.mod <==光碟档案系统模组

-rw-r--r--. lvm.mod <==LVM档案系统模组

-rw-r--r- -. mdraid09.mod <==软体磁碟阵列模组

-rw-r--r--. minix.mod <==MINIX相关档案系统模组

-rw-r--r--. msdospart.mod < ==一般MBR分割表

-rw-r--r--. part\_gpt.mod <==GPT分割表

-rw-r--r--. part\_msdos.mod <==MBR分割表

-rw-r-- r--. scsi.mod <==SCSI相关模组

-rw-r--r--. usb\_keyboard.mod <==底下两个为USB相关模组

-rw-r--r--. usb.mod

-rw-r--r--. vga.mod <==VGA显示卡相关模组

-rw-r--r--. xfs.mod <==XFS档案系统模组

从上面的说明你可以知道/boot/grub2/ 目录下最重要的就是(grub2.cfg) 以及各种文件系统的定义！我们的loader 读取了这种文件系统定义数据后，就能够认识文件系统并读取在该文件系统内的核心文件啰。

grub2的配置文件/boot/grub2/grub.cfg

gurb2优点

识别与支持较多的文件系统,并可以使用grub2的主程序直接在文件系统中查找内核文件

启动时可以(自行编辑与修改启动设置选项),类似bash

可以动态查找配置文件,而不需要在修改配置文件后重新安装grub2,即修改后下次启动就生效

磁盘与分区在grub2的代号

grub2最主要的功能是从磁盘中加载内核文件,所以grub2要能识别硬盘才行

(hd0,1) #一般的预设语法，由grub2自动判断分割格式

(hd0,msdos1) #此磁碟的分割为传统的MBR模式

(hd0,gpt1) #此磁碟的分割为GPT模式

够神了吧？跟/dev/sda1 风马牛不相干～怎么办啊？其实只要注意几个东西即可，那就是：

硬盘代号以小括号( ) 包起来；

硬盘以hd 表示，后面会接一组数字；

以『搜寻顺序』做为硬盘的编号！(这个重要！)

第一个搜寻到的硬盘为0 号，第二个为1 号，以此类推；

每颗硬盘的第一个partition 代号为1 ，依序类推。

/boot/grub2/grub.cfg配置文件(详细查找鸟哥,了解就行)

grub2配置文件维护/etc/default/grub 与/etc/grub.d

grub2主配置文件grub.cfg不建议修改,而是通过这两个文件来处理比较好

其它参考鸟哥

initramfs(虚拟文件系统)的重要性与建立新的initramfs

测试与安装grub2

启动前的额外功能修改

为个别选项设置密码

以上都都参考鸟哥即可

启动过程的问题解决

忘记root密码

旧版的linux只要能进入并挂载根目录,然后重设root密码即可

方法一

新版的systemd中,默认的rescue模式无法直接获取root权限,可以通过rd.reak的内核参数来处理,还需在chroot支持,有开selinux还要额外设定

重启后,进到启动画面,在可启动的选项按e进编辑模式,然后在linux16 那个内核项目后面加上re.break,然后ctrl + x保存，然后再重启

Generating "/run/initramfs/rdsosreport.txt"

Enter emergency mode. Exit the shell to continue.

Type "journalctl" to view system logs.

You might want to save "/run/initramfs/rdsosreport.txt" to a USB stick or /boot

after mounting them and attach it to a bug report.

switch\_root:/# #无须输入密码即可取得root权限！

switch\_root:/# mount #检查一下挂载点！一定会发现/sysroot才是对的！.....(前面省略).....

/dev/mapper/centos-root on /sysroot type xfs ( ro ,relatime,attr,inode64,noquota)

switch\_root:/# mount -o remount,rw /sysroot #要先让它挂载成可读写！

switch\_root:/# chroot /sysroot #实际切换了根目录的所在！取回你的环境了！

sh-4.2# echo "your\_root\_new\_pw" | passwd --stdin root

sh-4.2# touch /.autorelabel #很重要！变回SELinux的安全本文～

sh-4.2# exit

switch\_root:/# reboot

方法二

直接启动就以root执行bash方法

同样重启后在linux16 后加init=/bin/bash,这样启动后就获得一个bash(不需root密码但具有root权限),然后同样的remount根目录,再用echo passwd修改密码,然后重启即可(reboot没有,只能reset或强制关机后启动)

推荐方法1

因为文件系统错误而无法启动

比如/etc/fstab写错又没有mount -a测试,也只能输入root密码后获取bash 再重新挂载根目录再进行后续

不正常关机导至文件系统不一致也会和上面出现相同的问题

如果是扇区错误的情况

根据提示的文件系统类型错误来修复如xfs可用xfs\_repair,恢复不成功可将重要数据复制出来重新安装系统

基础系统设置与备份策略

网络设置

nmcli 通过命令来修改网络

修改主机名

显示主机名 hostnamectl

cat /etc/hostname

修改主机名

hostnamectl set-hostname 名称

日期与时间设置

timedatectl 显示是时间与时区

timedatectl list-timezone | grep -i new #查看是否支持newyouk时区

timedatectl set-timezone "America/New York" 设置为new your时区

timedatectl set-tme "2020-07-16 12:01" 设置时间,不需要再用hwclock去修正BIOS记录时间

用ntpdate手动网络校时

如果是用系统默认的自动校时则会启动协议相关的软件d,会多开几个Port,所以建议手动

ntpdate s2m.time.edu.cn #北京大学时间服务器

hwclock -w #手动将正确的时间写入BIOS

语系设置

一般我们用locale修改语系即可(目前软件bash语系)，修改LANG=es\_US.utf8, 永久就改配置文件/etc/locale.conf把LANG写进去

localectl 可以查看与设置系统语系,图形界面登录用到的就是系统语系

服务器硬件数据的收集

dmidecode 查看硬件设备

lspci 列出整个pc系统的pci接口设备

lsusb 列出目前系统上面各个USB端口状态,与连接的USB设备

smartctl 检测磁盘健康状态

软件安装, 源代码与Tarball

开放源码的软件安装与升级

如windows的安装软件,就是只能点下一步,我们并不知道软件里面的代码,万一软件有漏洞的话就只能等软件开发商提供补丁来修补

linux的源码源码安装软件的话,因为我们可以直接看源码,这样就可以提前修补软件漏洞,就更加安全,而且可以自定义软件的内容

只要有网络的存在,程序的漏洞就永远补不完,能补多少是多少

什么是开放源码,编译器与可执行文件

linux系统上真正识别的可执行文件其实是二进制程序,如/usr/bin/passwd /bin/touch

shell脚本只是利用bash程序提供的功能进行一些判断,而最终执行的除了bash提供的功能外,仍是调用了一些已经编译好的二进制程序,bash本身也是一个二进制程序

查看文件是否为二进制

[root@localhost ~]# file /bin/bash

/bin/bash: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.32, BuildID[sha1]=9223530b1aa05d3dbea7e72738b28b1e9d82fbad, stripped

You have new mail in /var/spool/mail/root

[root@localhost ~]# file /etc/init.d/network

/etc/init.d/network: Bourne-Again shell script, ASCII text executable

#如果是二进制且是可以执行是,会显示ELF 64-bit LSB executable, 同是会说明是否使用动态函数库(use shared libs)

#如果是一般脚本,会显示text executable

开放源代码

就是程序代码,写给人类看的程序,但机器不认识,所以无法执行

编译器

将程序代码转译成为机器看得懂的语言,类似翻译的角色

可执行文件

经过编译器变成的二进制程序,机器看得懂所以可以执行

如linux上的标准程序语言C,我们用C写好源代码好,用标准的编译器gcc来编译(可能会引用或调用其他的外部子程序或函数功能),产生一个可以执行的二进制程序

C源代通常以\*.c为扩展名,在编译的过程中还会产生所谓的目标文件,通常以\*.o扩展名

什么是函数库

如PAM提供的函数功能(验证用户的信息),我们可以在编写具有身份认证功能的程序时,直接引用该PAM的功能就好,而不需要重新设计认证机制

函数库

类类似子程序的角色,是可以被调用执行的一段功能函数

什么是make与configure

程序代码如果太多类似apache,那就要使用make命令来简化编译过程

执行make后,make会在当前目录下找makefile文本文件,而makefile里记录了源代码如何编译的信息,make会自动判别源代码变动情况来自动更新执行文件

通常软件开发商会写一个检测程序来检测安装软件的操作环境以及检测该环境是否有软件开发商所需要的功能,检测程序完毕后会主动建立一个makefile文件,通常检测程序文件名为configure或config

检测程序会检测的大概内容

是否有适合的编译器可编译本软件的源代码

是否已经存在本软件的需要的函数库或其它的依赖软件

操作系统平台是否适合软件,包括Linux的内核版本

内核的头文件(head include)是否存在(驱动程序必须要做的检测)

Tarball软件

就是一个打包与压缩好的软件包,通常为\*.tar.gz,里面的文件通常有

源你码文件

检测程序(configure或config)

本软件的简易说明与安装说明(INSTALL或README),安装时可以参考这两个文件

安装与升级

安装

tarball由厂商的网站下载

tarball解压,产生很多的源代码文件

开始以gcc进行源代码的编译(会产生目标文件object files)

然后以gcc进行函数库,主,子程序的链接,形成主要的二进制文件

将上述的二进制文件以及相关配置文件安装到自己的主机上面

为什么要升级

需要新的功能

旧版本存在安全缺陷

旧的版本性能不佳

升级分两大类,分别是

直接以源代码通过编译来安装与升级(tarball)

直接以编译好的二进制程序来安装与升级(rpm yum)

简单编译范例

编辑程序代码

[root@study ~]# vim hello.c <==用C语言写的扩展名建议用.c

#include <stdio.h>

int main(void)

{

printf("Hello World\n");

}

开始编译与执行

[root@study ~]# gcc hello.c

[root@study ~]# ll hello.c a.out

-rwxr-xr-x . 1 root root 8503 Sep 4 11:33 a.out <==此时会产生这个档名

-rw-r--r--. 1 root root 71 Sep 4 11:32 hello.c

[root@study ~]# ./a.out

Hello World <==呵呵！成果出现了！

#默认状态下,直接以gcc编译并且没加任何参数,执行文件名会自动设为a.out

如果是要产生目标文件

[root@study ~]# gcc -c hello.c

[root@study ~]# ll hello\*

-rw-r--r--. 1 root root 71 Sep 4 11:32 hello.c

-rw-r--r--. 1 root root 1496 Sep 4 11:34 hello.o <==就是被产生的目标档

[root@study ~]# gcc -o hello hello.o

[root@study ~]# ll hello\*

-rwxr-xr-x . 1 root root 8503 Sep 4 11:35 hello <==这就是可执行档！-o的结果

-rw-r--r--. 1 root root 71 Sep 4 11:32 hello.c

-rw-r--r--. 1 root root 1496 Sep 4 11:34 hello.o

[root@study ~]# ./hello

Hello World

主,子程序的链接，子程序的编译

编定主,子程序

# 1.编辑主程式：

[root@study ~]# vim thanks.c

#include <stdio.h>

int main(void)

{

printf("Hello World\n");

thanks\_2();

}

#上面的thanks\_2();那一行就是呼叫副程式啦！

[root@study ~]# vim thanks\_2.c

#include <stdio.h>

void thanks\_2(void)

{

printf("Thank you!\n");

}

进行程序的编译与链接

# 2.开始将原始码编译成为可执行的binary file ：

[root@study ~]# gcc -c thanks.c thanks\_2.c

[root@study ~]# ll thanks\*

-rw-r--r--. 1 root root 75 Sep 4 11:43 thanks\_2.c

-rw-r--r--. 1 root root 1496 Sep 4 11:43 thanks\_2.o <==编译产生的！

-rw-r--r--. 1 root root 91 Sep 4 11:42 thanks.c

-rw-r--r--. 1 root root 1560 Sep 4 11:43 thanks.o <==编译产生的！

[root@study ~]# gcc -o thanks thanks.o thanks\_2.o

[root@study ~]# ll thanks\*

-rwxr-xr-x. 1 root root 8572 Sep 4 11:44 thanks <==最终结果会产生这玩意儿

# 3.执行一下这个档案：

[root@study ~]# ./thanks

Hello World

Thank you!

由于源代码文件有时并非仅有一个文件,所以我们无法直接编译,需要先产生目标文件,然后再以链接制作为二进制文件,另外如果有一天thanks\_2文件要修改,修改thanks\_2c文件内容,则只需重新编译thanks\_2.c来产生新的thanks\_2.o,再以链接制作出二进制可执行文件,不必重新编译其它没修改过的源代码文件

调用外部函数库

[root@study ~]# vim sin.c

#include <stdio.h>

#include <math.h>

int main(void)

{

float value;

value = sin ( 3.14 / 2 );

printf("%f\n",value);

}

编译

[root@study ~]# gcc sin.c

#新的GCC会主动将函数调进来给你用，所以只要加上include <math.h>就好了！

实际上数学函数库使用的是libm.so这个函数库,所以最好在编译时将这个函数库包含进去,另外这个函数库放置的地方是系统默认的/lib 与/lib64,所以无需使用-L去加入查找的目录(除非函数库路径有改变),所以上面的编译过程可以是

[root@study ~]# gcc sin.c -lm -L/lib -L/lib64 <==重点在-lm

-l ：是『加入某个函式库(library)』的意思，

m ：则是libm.so 这个函式库，其中， lib 与副档名(.a 或.so)不需要写

gcc简易用法

#仅将原始码编译成为目标档，并不制作连结等功能：

[root@study ~]# gcc -c hello.c

#会自动的产生hello.o这个档案，但是并不会产生binary执行档。

#在编译的时候，依据作业环境给予最佳化执行速度

[root@study ~]# gcc -O hello.c -c

#会自动的产生hello.o这个档案，并且进行最佳化喔！

#在进行binary file制作时，将连结的函式库与相关的路径填入

[root@study ~]# gcc sin.c -lm -L/lib -I/usr/include

#这个指令较常下达在最终连结成binary file的时候，

# -lm 指的是libm.so 或libm.a 这个函式库档案；

# -L 后面接的路径是刚刚上面那个函式库的搜寻目录；

# -I 后面接的是原始码内的include 档案之所在目录。

#将编译的结果输出成某个特定档名

[root@study ~]# gcc -o hello hello.c

# -o后面接的是要输出的binary file档名

#在编译的时候，输出较多的讯息说明

[root@study ~]# gcc -o hello hello.c -Wall

#加入-Wall之后，程式的编译会变的较为严谨一点，所以警告讯息也会显示出来！

用make进行宏编译

make的功能就是用来简化编译过程所执行的命令

假设我的执行文件里面包含了四个原代码文件，分别是main.c haha​​.c sin\_value.c cos\_value.c 这四个源代码文件

# 1.先进行目标文件的编译，最终会有四个\*.o的档名出现：

[root@study ~]# gcc -c main.c

[root@study ~]# gcc -c haha.c

[ root@study ~]# gcc -c sin\_value.c

[root@study ~]# gcc -c cos\_value.c

# 2.再进行连结成为执行文件，并加入libm的数学函式，以产生main执行文件：

[root@study ~]# gcc -o main main.o haha.o sin\_value.o cos\_value.o -lm

# 3.本程式的执行结果，必须输入姓名、360度角的角度值来计算：

[root@study ~]# ./main

Please input your name: VBird <==这里先输入名字

Please enter the degree angle (ex> 90): 30 <==输入以360度角为主的角度

Hi, Dear VBird, nice to meet you. <==这三行为输出的结果喔！

The Sin is: 0.50

The Cos is: 0.87

编译的过程需要进行好多动作啊！而且如果要重新编译，则上述的流程得要重新来一遍，光是找出这些指令就够烦人的了！如果可以的话，能不能一个步骤就给他完成上面所有的动作呢？那就利用make 这个工具吧！

先建立makefile文件

# 1.先编辑makefile这个规则档，内容只要作出main这个执行档

[root@study ~]# vim makefile

main: main.o haha.o sin\_value.o cos\_value.o

gcc -o main main.o haha.o sin\_value.o cos\_value.o -lm

#注意：第二行的gcc之前是<tab>按键产生的空格喔！

# 2.尝试使用makefile制订的规则进行编译的行为：

[root@study ~]# rm -f main \*.o <==先将之前的目标档去除

[root@study ~]# make

cc -c -o main.o main.c

cc -c -o haha​​.o haha​​.c

cc -c -o sin\_value.o sin\_value.c

cc -c -o cos\_value.o cos\_value.c

gcc -o main main.o haha​​.o sin\_value.o cos\_value.o -lm

# 此时make 会去读取makefile 的内容，并根据内容直接去给他编译相关的档案啰！

# 3.在不删除任何档案的情况下，重新执行一次编译的动作：

[root@study ~]# make

make: `main' is up to date.

# 看到了吧！是否很方便呢！只会进行更新(update) 的动作而已。

我们仅写出了main需要的目标文件,结果make会主动判断每个目标文件相关的源代码文件,并直接编译,最后再进行链接操作,真方便

所以,使用make的好处

简化编译

若编译完成后,修改了某个源代码文件,则make仅会针对被修改了的文件进行编译,其他的不被修改

最后可以依照依赖性更新(update)执行文件

makefile基本语法与变量

标的(target): 目标档1 目标档2

<tab> gcc -o 欲建立的执行档目标档1 目标档2

想要两个以上的操作,如执行一个命令就直接清除所有的目标文件与执行文件

# 1.先编辑makefile来建立新的规则，此规则的标的名称为clean ：

[root@study ~]# vi makefile

main: main.o haha​​.o sin\_value.o cos\_value.o

gcc -o main main.o haha​​.o sin\_value.o cos\_value.o -lm

clean:

rm -f main main.o haha​​.o sin\_value.o cos\_value.o

# 2.以新的标的(clean)测试看看执行make的结果：

[root@study ~]# make clean <==就是这里！透过make以clean为标的

rm -rf main main.o haha​​.o sin\_value.o cos\_value.o

用shell与变量简化

[root@study ~]# vi makefile

LIBS = -lm

OBJS = main.o haha​​.o sin\_value.o cos\_value.o

main: ${OBJS}

gcc -o main ${OBJS} ${LIBS}

clean:

rm -f main ${OBJS}

由于gcc在进行编译时，会主动的去读取CFLAGS这个环境变量，所以，你可以直接在shell定义出这个环境变量，也可以在makefile文件里面去定义，更可以在指令行当中给予这个咚咚呢！例如：

[root@study ~]# CFLAGS="-Wall" make clean main

#这个动作在上make进行编译时，会去取用CFLAGS的变数内容！

也可以这样：

[root@study ~]# vi makefile

LIBS = -lm

OBJS = main.o haha​​.o sin\_value.o cos\_value.o

CFLAGS = -Wall

main: ${OBJS}

gcc -o main ${OBJS} ${LIBS}

clean:

rm -f main ${OBJS}

其它详细参考鸟哥

Tarball管理与建议

tarball安装的基本步骤

取得原始文件：将tarball 文件在/usr/local/src 目录下解压缩；

取得步骤流程：进入新建立的目录底下，去查阅INSTALL 与README 等相关文件内容(很重要的步骤！)；

相依属性软体安装：根据INSTALL/README 的内容察看并安装好一些相依的软体(非必要)；

建立makefile：以自动检测程序(configure 或config) 检测操作环境，并建立Makefile 这个文件；

编译：以make 这个程序并使用该目录下的Makefile 做为他的参数配置文件，来进行make (编译或其他) 的动作；

安装：以make 这个程序，并以Makefile 这个参数文件，依据 install 这个目标的(target) 的指定来安装到正确的路径！

makefile制处出来后,里面会有相当多的target,常见的是install与clean,make clean代表将目标文件清除掉,make则是将源代码进行编译而已,注意,编译完成的可执行文件与相关配置文件还在源代码所在的目录中,因此最后要进行make install来将编译完成的所有东西都安装到正确的路径中,这样就可以使用该软件

命令执行顺序

./configure

检测程序

make clean

不一定要执行这个,但是希望执行一下,因为不确定里面到底有没有上次编译过的目标文件(\*.og)

make

make install

将上一步make所编译完成的内容安装到预定的目录中,从而完成安装

ntp tarball安装示范

[root@study ~]# cd /usr/local/src <==切换目录

[root@study src]# tar -zxvf /root/ntp-4.2.8p3.tar.gz <==解压缩到此目录

ntp -4.2.8p3/ <==会建立这个目录喔！

ntp-4.2.8p3/CommitLog

....(底下省略)....

[root@study src]# cd ntp-4.2.8p3

[root@study ntp-4.2.8p3]# vi INSTALL <==记得README也要看一下！

#特别看一下28行到54行之间的安装简介！可以了解如何安装的流程喔！

检查configure 支援参数，并实际建置makefile 规则档

[root@study ntp\*]# ./configure --help | more <==查询可用的参数有哪些

--prefix=PREFIX install architecture-independent files in PREFIX

--enable-all-clocks + include all suitable non-PARSE clocks:

--enable-parse-clocks - include all suitable PARSE clocks:

# 上面列出的是比较重要的，或者是你可能需要的参数功能！

[root@study ntp\*]# ./configure --prefix=/usr/local/ntp \

> --enable-all-clocks --enable-parse-clocks <==开始建立makefile

checking for a BSD-compatible install... /usr/bin/install -c

checking whether build environment is sane... yes

....(中间省略)....

checking for gcc... gcc <==也有找到gcc编译器了！

....(中间省略)....

config.status: creating Makefile <==现在知道这个重要性了吧？

config.status: creating config.h

config.status: creating evconfig-private.h

config.status: executing depfiles commands

config.status: executing libtool commands

一般来说configure设定参数较重要的就是那个--prefix=/path了，--prefix后面接的路径就是『这个软体未来要安装到那个目录去？』如果你没有指定--prefix=/path这个参数，通常预设参数就是/usr/local至于其他的参数意义就得要参考./configure --help了！这个动作完成之后会产生makefile或Makefile这个档案。当然啦，这个侦测检查的过程会显示在萤幕上， 特别留意关于gcc的检查，还有最重要的是最后需要成功的建立起Makefile才行！

最后开始编译与安装噜！

[root@study ntp\*]# make clean; make

[root@study ntp\*]# make check

[root@study ntp\*]# make install

#将资料给他安装在/usr/local/ntp底下

利用patch更新源代码

测试旧版程式的功能

[root@study ~]# tar -zxvf main-0.1.tgz

[root@study ~]# cd main-0.1

[root@study main-0.1]# make clean main

[root@study main-0.1]# ./main

version 0.1

Please input your name: VBird

Please enter the degree angle (ex> 90): 45

Hi, Dear VBird, nice to meet you.

The Sin is: 0.71

The Cos is: 0.71

查看patch file内容

[root@study main-0.1]# vim ~/main\_0.1\_to\_0.2.patch

diff -Naur main-0.1/cos\_value.c main-0.2/cos\_value.c

--- main-0.1/cos\_value.c 2015-09-04 14:46:59.200444001 +0800

+++ main-0.2/cos\_value.c 2015-09-04 14:47:10.215444000 +0800

@@ -7,5 +7,5 @@

{

float value;

....(底下省略)....

patch 的基本语法如下：

patch -p数字< patch\_file

特别留意那个『 -p数字』，那是与patch\_file 里面列出的档名有关的资讯。假如在 patch\_file 第一行写的是这样：

\*\*\* /home/guest/example/expatch.old

那么当我下达『 patch -p0 < patch\_file 』时，则更新的档案是『 /home/guest/example/expatch.old 』，如果『 patch -p1 < patch\_file』，则更新的档案为『home/guest/ example/expatch.old』，如果『patch -p4 < patch\_file』则更新『expatch.old』，也就是说， -pxx那个xx代表『拿掉几个斜线(/)』的意思！这样可以理解了吗？好了，根据刚刚上头的资料，我们可以发现比较的档案是在main-0.1/xxx与main-0.2/xxx ，所以说，如果你是在main-0.1底下，并且想要处理更新时，就得要拿掉一个目录(因为并没有main-0.2的目录存在，我们是在当前的目录进行更新的！)，因此使用的是-p1才对喔

更新原始码，并且重新编译程式！

[root@study main-0.1]# patch -p1 < ../main\_0.1\_to\_0.2.patch

patching file cos\_value.c

patching file main.c

patching file Makefile

patching file sin\_value.c

#请注意，鸟哥目前所在目录是在main-0.1底下喔！注意与patch档案的相对路径！

# 虽然有五个档案，但其实只有四个档案有修改过喔！上面显示有改过的档案！

[root@study main-0.1]# make clean main

[root@study main-0.1]# ./main

version 0.2

Please input your name: VBird

Please enter the degree angle (ex> 90): 45

Hi, Dear VBird, nice to meet you.

The sin(45.000000) is: 0.71

The cos(45.000000) is: 0.71

#你可以发现，输出的结果中版本变了，输出资讯多了括号()喔！

[root@study main-0.1]# make install <==将他安装到/usr/local/bin给大家用

cp -a main /usr/local/bin

[root@study main-0.1]# main <==直接输入指令可执行！

[root@study main-0.1]# make uninstall <==移除此软体！

rm -f /usr/local/bin/main

很有趣的练习吧！所以你只要下载patch file就能够对你的软体原始码更新了！只不过更新了原始码并非软体就更新！你还是得要将该软体进行编译后，才会是最终正确的软体喔！因为patch的功能主要仅只是更新原始码档案而已！切记切记！此外，如果你patch错误呢？没关系的！我们的patch是可以还原的啊！透过『 patch -R < ../main\_0.1\_to\_0.2.patch 』就可以还原啦！很有趣吧！

函数库管理

软件之间会互相使用彼此提供的函数库来使用其特殊的功能,如需要验证身份的程序都习惯利用PAM这个模块提供的验证机制来实践,网络连接习惯利用SSL函数库来实现连接加密的机制

函数库根据被使用的类型而分为静态(static)与动态(Dynamic)函数库

静态函数库

扩展名

libxxx.a

编译操作

这类函数库在编译的时候会直接整合到执行程序中,所以这类编译成的文件会比较大

独立执行的状态

这类函数库最大的优点,编译成功的可执行文件可以独立运行,而不需要再向外部要求读取函数库的内容

升级难易度

因为函数库是直接整合到执行文件中,所以函数库升级时,整个执行文件必须要重新编译才能将新版的函数库整合到程序中

动态函数库

扩展名

libxxx.so

编译操作

动态函数库在编译的时候,在程序里面只有一个指针(pinter)的位置而已,就是说动态函数库的内容并没有被整合到执行文件当中,而是执行文件要使用到函数库的功能时,程序才会去读取函数库来使用,所以它的文件较小

独立执行的状态

不能独立执行,因为当使用到函数库功能时,程序才会去读取函数库,所以函数库文件必须要存在(目录也不能改变)

升级难易度

函数库升级后,执行文件根本不需要进行重新编译的操作,因为执行文件会直接指向新的函数库文件(前提是函数库新旧版本文件名相同)

目前linux发行版比较倾向于使用动态函数库

ldconfig 与 /etc/ld.so.conf

将常用到的动态函数库加载到内存当中(缓存,cache),如此一来,当软件要使用动态函数库时,就不需要从头在硬盘读取了,提高了读取速度

动态函数库加载到内存

先在/etc/ld.so.conf里面写入(目录)

利用ldconfig执行文件将/etc/ld.so.conf的数据读入缓存中

同时将数据记录一份至/etc/ld.so.cache

[root@study ~]# ldconfig [-f conf] [ -C cache]

[root@study ~]# ldconfig [-p]

选项与参数：

-f conf ：那个conf 指的是某个档案名称，也就是说，使用conf 作为libarary

函式库的取得路径，而不以/etc/ld.so.conf 为预设值

-C cache：那个cache 指的是某个档案名称，也就是说，使用cache 作为快取暂存

的函式库资料，而不以/etc/ld.so.cache 为预设值

-p ：列出目前有的所有函式库资料内容(在/etc/ld.so.cache 内的资料！)

范例一：假设我的Mariadb资料库函式库在/usr/lib64/mysql当中，如何读进cache ？

[root@study ~]# vim /etc/ld.so.conf.d/vbird.conf

/usr/lib64/mysql <==这一行新增的啦！

[root@study ~]# ldconfig <==画面上不会显示任何的资讯，不要太紧张！正常的！

[root@study ~]# ldconfig -p

924 libs found in cache `/etc/ld.so.cache'

p11-kit-trust.so (libc6,x86-64) => /lib64/p11-kit-trust.so

libzapojit-0.0.so.0 (libc6,x86-64) => /lib64/libzapojit-0.0.so.0

....(底下省略)....

#函式库名称=>该函式库实际路径

可以参考上面的例子把其它的需要的动态函数加入缓存就行

程序的动态函数库解析 ldd

[root@study ~]# ldd [-vdr] [filename]

选项与参数：

-v ：列出所有内容资讯；

-d ：重新将资料有遗失的link 点秀出来！

-r ：将ELF 有关的错误内容秀出来！

范例一：找出/usr/bin/passwd这个档案的函式库资料

[root@study ~]# ldd /usr/bin/passwd

....(前面省略)....

libpam.so.0 = > /lib64/libpam.so.0 (0x00007f5e683dd000) <==PAM模组

libpam\_misc.so.0 => /lib64/libpam\_misc.so.0 (0x00007f5e681d8000)

libaudit.so.1 => /lib64/libaudit.so.1 (0x00007f5e67fb1000) <==SELinux

libselinux.so.1 => /lib64/libselinux.so.1 (0x00007f5e67d8c000) <==SELinux

....(底下省略)....

#我们前言的部分不是一直提到passwd有使用到pam的模组吗！怎么知道？

# 利用ldd 察看一下这个档案，看到libpam.so 了吧？这就是pam 提供的函式库

范例二：找出/lib64/libc.so.6这个函式的相关其他函式库！

[root@study ~]# ldd -v /lib64/libc.so.6

/lib64/ld-linux-x86-64.so.2 (0x00007f7acc68f000)

linux-vdso.so.1 => (0x00007fffa975b000)

Version information: <==使用-v选项，增加显示其他版本资讯！

/lib64/libc.so.6:

ld-linux-x86-64.so.2 (GLIBC\_2.3) => /lib64/ld-linux-x86-64.so.2

ld-linux-x86-64.so.2 (GLIBC\_PRIVATE) => /lib64/ld-linux-x86-64.so.2

ldd可以查看相关的依赖

校验软件正确性

[root@study ~]# md5sum/sha1sum/sha256sum [-bct] filename

[root@study ~]# md5sum/sha1sum/sha256sum [--status|--warn] --check filename

选项与参数：

-b ：使用binary 的读档方式，预设为Windows/DOS 档案型态的读取方式；

-c ：检验档案指纹；

-t ：以文字型态来读取档案指纹。

范例一：将刚刚的档案下载后，测试看看指纹码

[root@study ~]# md5sum ntp-4.2.8p3.tar.gz

b98b0cbb72f6df04608e1dd5f313808b ntp-4.2.8p3.tar.gz

# 看！显示的编码是否与上面相同呢？赶紧测试看看！

md5sum 得到的字串与官网提供的md5字串对比一样就说明下载的软件没有被修改过,安全

额外应用

如可以给下面文件做md5值记录

/etc/passwd

/etc/shadow (假如你不让使用者改密码了)

/etc/group

/usr/bin/passwd

/sbin/rpcbind

/bin/login (这个也很容易被骇！)

/bin/ls

/bin/ps

/bin/top

上面这些文件最容易被木马或黑客修改,所以可以将这些文件的md5校验值记录下来,以shell脚本来自行检查是否不同来确保安全

软件安装RPM,SRPM与YUM

源代码虽然可以定制,但是需要检测环境,设置编译参数,编译,最后还要依据个人喜好安装特定位置,过程是真麻烦

如果我们的linux系统与厂商一样,那么在厂商的系统上面编译出来的执行文件,自然就能在我们的系统上面执行,就是说厂商先在他们的系统上面编译好了我们用户所需要的软件,然后将这个编译好的可执行的软件直接发布给用户来安装

厂商将编译好的软件所有相关的文件打包成一个特珠格式的文件,这个软件安装文件内还包含了预先检测系统与依赖软件的脚本,客户端获取这个文件后,通过特定的命令来安装,那么该文件就会依照内部的脚本来检测依赖的辅助软件是否存在,符合就开始安装

目前Linux常见软件安装方式有两种

dpkg

ubuntu等

rpm

red hat, centos等

rpm与srpm

rpm就是将要安装的软件先编译过,且打包成为rpm机制的文件,通过打包好的软件的默认数据库,记录软件要安装的必须具备的依赖性软件,当在你的Linux安装时,rpm会先依照软件里面的数据查询linux依赖属性是否满足,满足则安装,安装时将软件的信息写入rpm数据库,以便未来查询验证与反安装

优点

由于已编译且打包完毕,所以软件传输与安装方重(不需要重新编译)

软件信息都记录在Linux主机上的数据库,方便查询升级与反安装

缺点

软件安装的环境必须与打包时的环境需求一致或相当

需要满足软件的依赖属性需求

反安装时,最底层的软件不需先删除,否则可能造成系统问题

通常不同的Linux的发行版所发布的rpm文件,并不能用在其他的Linux版本上,那只能用srpm

srpm

source rpm的意思,就是这个RPM文件里面含有源代码(SRPM所提供的软件内空并没有经过编译,提供的是源代码)

通常扩展名 \*\*\*.src.rpm

如何安装

先将该软件以RPM管理的方式编译,此时SRPM会被编译成为RPM文件

然后将编译完成的RPM文件安装到LINUX系统中

为什么不用tarball呢？

因为SRPM虽然内容是源代码,但是它仍然含有该软件所需要的依赖性软件说明以及所有RPM文件所提供的数据,同时与RPM不同的是,它也提供了参数配置文件(configure与makefile)

用处

因为rpm文件必须要在相同的Linux环境下才能安装,而SRPM是源代码的格式,我们就可以通过修改SRPM内的参数配置文件,然后重新编译产生能适合我们LINUX环境的RPM文件

什么是i386, i586, i686, noarch, x86\_64

从上面的说明，现在我们知道RPM 与SRPM 的格式分别为：

xxxxxxxxx.rpm <==RPM的格式，已经经过编译且包装完成的rpm档案；

xxxxx.src.rpm <==SRPM的格式，包含未编译的原始码资讯。

那么我们怎么知道这个软体的版本、适用的平台、编译释出的次数呢？只要透过档名就可以知道了！例如 rp-pppoe-3.11-5.el7.x86\_64.rpm 这的档案的意义为：

rp-pppoe - 3.11 - 5 .el7.x86\_64 .rpm

软体名称 软体的版本 发布的次数 适合的硬体平台 扩展名

i386 i586 i686 都是比较旧的CPU等级

X86\_64 针对64位的CPU进行优化编译设置

noarch 就是没有任何硬件等级上的限制,一般来说这类rpm文件里面应该没有二进制执行程序,属于shell脚本方面的软件多

安装软件最好与硬件平台搭配,因为软件有针对CPU硬件平台进行过参数优化

RPM属性依赖的解决方式 YUM在线升级

为了重复利有即有软件功能,很多软件都会以函数库的方式发布部分功能,此外为了节省用户数据量,目前linux发行版在发布软件时,都会将软件的内容分为一般使用与开发使用(development)两大类.如pam-x.x.rpm与pam-devel-x.x.rpm,而默认情况下,大部分的software-devel-x.x.rpm都不会安装

因为上述现象,所以rpm文件就会有所谓的属性依赖的问题,可用 yum机制 来解决

先将发布的软件放到YUM服务器,然后分析这些软件的依赖属性问题

将软件内记录信息记录下来(header),再将这些信息分析后记录成软件相关的列表(包含软件依赖属性与软件的URL即下载地址),这些软件与列表数据所在的本机或网络上的位置就称为软件源或轮件他库(repository)

当客户端有软件安装后,客户端主机主动向YUM服务器的软件源地址下载列表(会放到/var/cache/yum),然后通过列表的数据与本机RPM数据库已存在的软件比较,就能一口气下载

所需具有依赖属性的软件,下载软件后再用rpm安装

RPM软件管理程序

其实通过YUM就可以安装软件了,rpm只剩下查询与检验功能的多

RPM默认安装路径

软件的相关信息会写入/var/lib/rpm下的数据库文件中,未来软件升级版本比较与软件查询,RPM数字签名信息都是记录在这里,不能被删除

RPM安装

rpm -ivh packege.rpm

rpm -ivh packege.rpm egegegeg.rpm 一次安装两个

rpm -ivh packege.rpm --nodeps #忽视依赖,不建议,一般就-ivh参数即可

--replacepkgs #重新安装软件,如软件的某个重要文件修改错了或丢失

RPM升级与更新

rpm

-Uvh

后面接的软件即使没有安装,系统也将直接安装,若本身软件安装过,则系统自动更新到新版本

-Fvh

后面接的软件如果没有安装,则该软件不会安装.只有软件本身有安装,才给予升级

建议 -Fvh 来升级安装,如果软件没有安装过需要安装,可通过-ivh来安装,不过现在有yum了,这些方法就都不用了

RPM查询(query)

[root@study ~]# rpm -qa <==已安装软件

[root@study ~]# rpm -q[licdR]已安装的软件名称 <==已安装软件

[root@study ~]# rpm -qf存在于系统上面的某个档名 <==已安装软件

[root@study ~]# rpm -qp[licdR]未安装的某个文件名称 <==查阅RPM文件

选项与参数：

查询已安装软件的资讯：

-q ：仅查询，后面接的软件名称是否有安装；

-qa ：列出所有的，已经安装在本机Linux 系统上面的所有软件名称；

-qi ：列出该软件的详细资讯(information)，包含开发商、版本与说明等；

-ql ：列出该软件所有的文件与目录所在完整档名(list)；

-qc ：列出该软件的所有设定档(找出在/etc/ 底下的档名而已)

-qd ：列出该软件的所有说明档(找出与man 有关的文件而已)

-qR ：列出与该软件有关的相依软件所含的文件(Required 的意思)

-qf ：由后面接的文件名称，找出该文件属于哪一个已安装的软件；

-q --scripts：列出是否含有安装后需要执行的脚本档，可用以debug 喔！

查询某个RPM 文件内含有的资讯：

-qp[icdlR]：注意-qp 后面接的所有参数以上面的说明一致。但用途仅在于找出

某个RPM 文件内的资讯，而非已安装的软件资讯！注意！

范例一：找出你的Linux是否有安装logrotate这个软体？

[root@study ~]# rpm -q logrotate

logrotate-3.8.6-4.el7.x86\_64

[root@study ~]# rpm -q logrotating

package logrotating is not installed

# 注意到，系统会去找是否有安装后面接的软体名称。注意，不必要加上版本喔！

# 至于显示的结果，一看就知道有没有安装啦！

范例二：列出上题当中，属于该软体所提供的所有目录与档案：

[root@study ~]# rpm -ql logrotate

/etc/cron.daily/logrotate

/etc/logrotate.conf

....(以下省略)....

#可以看出该软体到底提供了多少的档案与目录，也可以追踪软体的资料。

范例三：列出logrotate这个软体的相关说明资料：

[root@study ~]# rpm -qi logrotate

Name : logrotate #软体名称

Version : 3.8.6 #软体的版本

Release : 4.el7 #释出的版本

Architecture : x86\_64 #编译时所针对的硬体等级

Install Date: Mon 04 May 2015 05:52:36 PM CST #这个软体安装到本系统的时间

Group : System Environment/Base #软体是放再哪一个软体群组中

Size : 102451 #软体的大小

License : GPL+ #释出的授权方式

Signature : RSA/SHA256, Fri 04 Jul 2014 11:34:56 AM CST, Key ID 24c6a8a7f4a80eb5

Source RPM : logrotate-3.8.6-4.el7.src.rpm #这就是SRPM的档名

Build Date : Tue 10 Jun 2014 05:58:02 AM CST #软体编译打包的时间

Build Host : worker1.bsys. centos.org #在哪一部主机上面编译的

Relocations : (not relocatable)

Packager : CentOS BuildSystem <http://bugs.centos.org>

Vendor : CentOS

URL : https://fedorahosted.org/logrotate/

Summary : Rotates, compresses, removes and mails system log files

Description : #这个是详细的描述！

The logrotate utility is designed to simplify the administration of

log files on a system which generates a lot of log files. Logrotate

allows for the automatic rotation compression, removal and mailing of

log files. Logrotate can be set to handle a log file daily, weekly,

monthly or when the log file gets to a certain size. Normally,

logrotate runs as a daily cron job.

Install the logrotate package if you need a utility to deal with the

log files on your system.

# 列出该软体的information (资讯)，里面的资讯可多著呢，包括了软体名称、

# 版本、开发商、SRPM档案名称、打包次数、简单说明资讯、软体打包者、

# 安装日期等等！如果想要详细的知道该软体的资料，用这个参数来了解一下

范例四：分别仅找出logrotate的设定档与说明档

[root@study ~]# rpm -qc logrotate

[root@study ~]# rpm -qd logrotate

范例五：若要成功安装logrotate ，他还需要什么档案的帮忙？

[root@study ~]# rpm -qR logrotate

/bin/sh

config(logrotate) = 3.8.6-4.el7

coreutils >= 5.92

....(以下省略)....

# 由这里看起来，呵呵～还需要很多档案的支援才行喔！

范例六：由上面的范例五，找出/bin/sh是那个软体提供的？

[root@study ~]# rpm -qf /bin/sh

bash-4.2.46-12.el7.x86\_64

# 这个参数后面接的可是『档案』呐！不像前面都是接软体喔！

# 这个功能在查询系统的某个档案属于哪一个软体所有的。

范例七：假设我有下载一个RPM档案，想要知道该档案的需求档案，该如何？

[root@study ~]# rpm -qpR filename.i386.rpm

#加上-qpR ，找出该档案需求的资料！

常见的查询就是这些了！要特别说明的是，在查询本机上面的RPM 软体相关资讯时， 不需要加上版本的名称，只要加上软体名称即可！因为他会由/var/lib/rpm 这个资料库里面去查询， 所以我们可以不需要加上版本名称。但是查询某个RPM 档案就不同了，我们必须要列出整个档案的完整档名才行

RPM验证

作用方式是使用/var/lib/rpm下面的数据库内容来比对目前linux系统的环境下的所有安装文件

[root@study ~]# rpm -Va

[root@study ~]# rpm -V已安装的软体名称

[root@study ~]# rpm -Vp某个RPM档案的档名

[root@study ~]# rpm -Vf在系统上面的某个档案

选项与参数：

-V ：后面加的是软体名称，若该软体所含的档案被更动过，才会列出来；

-Va ：列出目前系统上面所有可能被更动过的档案； #可查看是否被攻击修改文件了

-Vp ：后面加的是档案名称，列出该软体内可能被更动过的档案；

-Vf ：列出某个档案是否被更动过～

范例一：列出你的Linux内的logrotate这个软体是否被更动过？

[root@study ~]# rpm -V logrotate

#如果没有出现任何讯息，恭喜你，该软体所提供的档案没有被更动过。

# 如果有出现任何讯息，才是有出现状况啊！

范例二：查询一下，你的/etc/crontab是否有被更动过？

[root@study ~]# rpm -Vf /etc/crontab

.......T. c /etc/crontab

# 瞧！因为有被更动过，所以会列出被更动过的资讯类型！

请修改/etc/logrotate.conf 内的rotate 变成 5

[root@study ~]# rpm -V logrotate

..5....T. c /etc/logrotate.conf

最前面的几个资讯是：

S ：(file Size differs) 档案的容量大小是否被改变

M ：(Mode differs) 档案的类型或档案的属性(rwx) 是否被改变？如是否可执行等参数已被改变

5 ：(MD5 sum differs) MD5 这一种指纹码的内容已经不同

D ：(Device major/minor number mis-match) 装置的主/次代码已经改变

L ：(readLink(2) path mis-match) Link 路径已被改变

U ：(User ownership differs) 档案的所属人已被改变

G ：(Group ownership differs) 档案的所属群组已被改变

T ：(mTime differs) 档案的建立时间已被改变

P ：(caPabilities differ) 功能已经被改变

至于那个c 代表的是『 Config file 』的意思，也就是档案的类型，档案类型有底下这几类：

c ：设定档(config file)

d ：文件资料档(documentation)

g ：鬼档案～通常是该档案不被某个软体所包含，较少发生！(ghost file)

l ：授权档案(license file)

r ：读我档案(read me)

配置文件被修改是正常,如果是二进制文件被修改就要小心了

RPM 数字签名

tarball可以使用md5校验值来检查文件,不过校验值也可能被修改,rpm使用数字签名来检验软件的来源

软件开发商推出的软件也会有自己的签名系统,厂商以数字签名系统产生一个专属于该软件的签名,并将该签名的公钥发出,当你要安装一个RPM文件时

首先必须先安装原厂发布的公钥文件

安装RPM时,RPM命令会读取RPM文件的签名信息与本机系统内的签名信息比对

签名相同则安装,否则警告停止安装

CENTOS 使用的数字签名系统为GNU计划的GPG(自己理解厂商发布的公钥文件里面包含有),Centos的数字签名位于

ll /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

透过底下的方式来安装即可喔！

[root@study ~]# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7 #系统本身有安装,了解即可

RPM反安装与重建数据库

rpm -e 软件名 #卸载软件

注意由于软件有依赖性,所以要卸载的软件应该由底层开始卸载,如果强制卸载软件,但软件本身有依赖性,则使用到某个函数库的软件可能功能会失效

重建数据库

由于软件的安装,升级与卸载等,某些操作或许可能会导致RPM数据库/var/lib/rpm内的文件损坏,可以重建一下数据库

rpm --rebuilddb

yum 查询 安装 升级 删除

yum查询

[root@study ~]# yum [option] [查询工作项目] [相关参数]

选项与参数：

[option]：主要的选项，包括有：

-y ：当yum 要等待使用者输入时，这个选项可以自动提供yes 的回应；

--installroot=/some/path ：将该软体安装在/some/path 而不使用预设路径

[查询工作项目] [相关参数]：这方面的参数有：

search ：搜寻某个软体名称或者是描述(description) 的重要关键字；

list ：列出目前yum 所管理的所有的软体名称与版本，有点类似rpm -qa；

info ：同上，不过有点类似rpm -qai 的执行结果；

provides：从档案去搜寻软体！类似rpm -qf 的功能！

范例一：搜寻磁碟阵列(raid)相关的软体有哪些？

[root@study ~]# yum search raid

Loaded plugins: fastestmirror, langpacks # yum系统自己找出最近的yum server

Loading mirror speeds from cached hostfile #找出速度最快的那一部yum server

\* base: ftp.twaren .net #底下三个软体库，且来源为该伺服器！

\* extras: ftp.twaren.net

\* updates: ftp.twaren.net

....(前面省略)....

dmraid-events-logwatch.x86\_64 : dmraid logwatch-based email reporting

dmraid-events.x86\_64 : dmevent\_tool (Device-mapper event tool) and DSO

iprutils.x86\_64 : Utilities for the IBM Power Linux RAID adapters

mdadm.x86\_64 : The mdadm program controls Linux md devices (software RAID arrays)

....(后面省略)....

# 在冒号(:) 左边的是软体名称，右边的​​则是在RPM 内的name 设定(软体名)

# 瞧！上面的结果，这不就是与RAID 有关的软体吗？如果想了解mdadm 的软体内容呢？

范例二：找出mdadm这个软体的功能为何

[root@study ~]# yum info mdadm

Installed Packages <==这说明该软体是已经安装的了

Name : mdadm <==这个软体的名称

Arch : x86\_64 <= =这个软体的编译架构

Version : 3.3.2 <==此软体的版本

Release : 2.el7 <==释出的版本

Size : 920 k <==此软体的档案总容量

Repo : installed <==软体库回报说已安装的

From repo : anaconda

Summary : The mdadm program controls Linux md devices (software RAID arrays)

URL : http://www.kernel.org/pub/linux/utils/raid/mdadm/

License : GPLv2+

Description : The mdadm program is used to create, manage, and monitor Linux MD (software

: RAID) devices. As such, it provides similar functionality to the raidtools

: package. However, mdadm is a single program, and it can perform

: almost all functions without a configuration file, though a configuration

: file can be used to help with some common tasks.

# 不要跟我说，上面说些啥？自己找字典翻一翻吧！拜托拜托！

范例三：列出yum伺服器上面提供的所有软体名称

[root@study ~]# yum list

Installed Packages <==已安装软体

GConf2.x86\_64 3.2.6-8.el7 @anaconda

LibRaw.x86\_64 0.14.8-5.el7.20120830git98d925 @base

ModemManager.x86\_64 1.1.0-6.git20130913.el7 @anaconda

....(中间省略)....

Available Packages <==还可以安装的其他软体

389-ds-base.x86\_64 1.3.3.1-20.el7\_1 updates

389-ds-base-devel.x86\_64 1.3.3.1-20.el7\_1 updates

389-ds-base-libs.x86\_64 1.3.3.1-20.el7\_1 updates

....(底下省略)....

# 上面提供的意义为：『 软体名称版本在那个软体库内』

范例四：列出目前伺服器上可供本机进行升级的软体有哪些？

[root@study ~]# yum list updates <==一定要是update s喔！

Updated Packages

NetworkManager.x86\_64 1:1.0.0-16.git20150121.b4ea599c.el7\_1 updates

NetworkManager-adsl.x86\_64 1:1.0.0-16.git20150121.b4ea599c.el7\_1 updates

....(底下省略)....

# 上面就列出在那个软体库内可以提供升级的软体与版本！

范例五：列出提供passwd这个档案的软体有哪些

[root@study ~]# yum provides passwd

passwd-0.79-4.el7.x86\_64 : An utility for setting or changing passwords using PAM

Repo : base

passwd-0.79-4.el7.x86\_64 : An utility for setting or changing passwords using PAM

Repo : @anaconda

# 找到啦！就是上面的这个软体提供了passwd 这个程式！

yum 安装与升级

[root@study ~]# yum [option] [安装与升级的工作项目] [相关参数]

选项与参数：

install ：后面接要安装的软体！

update ：后面接要升级的软体，若要整个系统都升级，就直接update 即可

范例一：将前一个练习找到的未安装的pam-devel安装起来

[root@study ~]# yum install pam-devel

Loaded plugins: fastestmirror, langpacks #首先的5行在找出最快的yum server

Loading mirror speeds from cached hostfile

\* base: ftp.twaren.net

\* extras: ftp.twaren.net

\* updates: ftp.twaren.net

Resolving Dependencies #接下来先处理『属性相依』的软体问题

--> Running transaction check

---> Package pam-devel.x86\_64 0:1.1.8-12.el7\_1.1 will be installed

--> Processing Dependency: pam(x86-64) = 1.1.8-12.el7\_1.1 for package: pam-devel-

1.1.8-12.el7\_1.1.x86\_64

--> Running transaction check

---> Package pam.x86\_64 0:1.1.8-12.el7 will be updated

---> Package pam.x86\_64 0:1.1.8-12.el7\_1.1 will be an update

--> Finished Dependency Resolution

Dependencies Resolved

# 由上面的检查发现到pam 这个软体也需要同步升级，这样才能够安装新版pam-devel 喔！

# 至于底下则是一个总结的表格显示！

================================================== ========================================

Package Arch Version Repository Size

================================================== ========================================

Installing:

pam-devel x86\_64 1.1.8-12.el7\_1.1 updates 183 k

Updating for dependencies:

pam x86\_64 1.1.8-12.el7\_1.1 updates 714 k

Transaction Summary

================================================== ========================================

Install 1 Package #要安装的是一个软体

Upgrade ( 1 Dependent package) #因为相依属性问题，需要额外加装一个软体！

Total size: 897 k

Total download size: 183 k #总共需要下载的容量！

Is this ok [y/d/N]: y #你得要自己决定是否要下载与安装！当然是y啊！

Downloading packages: #开始下载啰！

warning: /var/cache/yum/x86\_64/7/updates/packages/pam-devel-1.1.8-12.el7\_1.1.x86\_64.rpm:

Header V3 RSA/SHA256 Signature, key ID f4a80eb5: NOKEY

Public key for pam-devel-1.1.8-12.el7\_1.1.x86\_64.rpm is not installed

pam-devel-1.1.8-12.el7\_1.1.x86\_64.rpm | 183 kB 00:00:00

Retrieving key from file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

Importing GPG key 0xF4A80EB5:

Userid : "CentOS-7 Key (CentOS 7 Official Signing Key) <security@centos.org>"

Fingerprint: 6341 ab27 53d7 8a78 a7c2 7bb1 24c6 a8a7 f4a8 0eb5

Package : centos-release-7-1.1503.el7.centos.2.8.x86\_64 (@anaconda)

From : /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

Is this ok [y/N]: y #只有在第一次安装才会出现这个项目『确定要安装数位签章』才能继续！

Running transaction check

Running transaction test

Transaction test succeeded

Running transaction

Warning: RPMDB altered outside of yum.

Updating : pam-1.1.8-12.el7\_1.1.x86\_64 1/3

Installing : pam-devel-1.1.8-12.el7\_1.1.x86\_64 2/3

Cleanup : pam-1.1.8-12.el7.x86\_64 3/3

Verifying : pam-1.1.8-12.el7\_1.1.x86\_64 1/3

Verifying : pam-devel-1.1.8-12.el7\_1.1.x86\_64 2/3

Verifying : pam-1.1.8-12.el7.x86\_64 3/3

Installed:

pam-devel.x86\_64 0:1.1.8-12.el7\_1.1

Dependency Updated:

pam.x86\_64 0:1.1.8-12.el7\_1.1

yum 删除软件

[root@study ~]# yum remove pam-devel

Loaded plugins: fastestmirror, langpacks

Resolving Dependencies <==同样的，先解决属性相依的问题

--> Running transaction check

---> Package pam-devel.x86\_64 0:1.1.8-12.el7\_1.1 will be erased

--> Finished Dependency Resolution

Dependencies Resolved

================================================== ========================================

Package Arch Version Repository Size

================================================== ========================================

Removing:

pam-devel x86\_64 1.1.8-12.el7\_1.1 @updates 528 k

Transaction Summary

================================================== ========================================

Remove 1 Package #还好！没有相依属性的问题，仅移除一个软体！

Installed size: 528 k

Is this ok [y/N]: y

Downloading packages:

Running transaction check

Running transaction test

Transaction test succeeded

Running transaction

Erasing : pam-devel-1.1.8-12.el7\_1.1.x86\_64 1/1

Verifying : pam-devel-1.1.8-12.el7\_1.1.x86\_64 1/1

Removed:

pam-devel.x86\_64 0:1.1.8-12.el7\_1.1

Complete!

yum配置文件

[root@study ~]# vim /etc/yum.repos.d/CentOS-Base.repo

[base]

name=CentOS-$releasever - Base

mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os&infra=$infra

#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/

gpgcheck=1

gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

如上所示，鸟哥仅列出base 这个软体库内容而已，其他的软体库内容请自行查阅啰！上面的资料需要注意的是：

[base]：代表软体库的名字！中括号一定要存在，里面的名称则可以随意取。但是不能有两个相同的软体库名称， 否则yum 会不晓得该到哪里去找软体库相关软体清单档案。

name：只是说明一下这个软体库的意义而已，重要性不高！

mirrorlist=：列出这个软体库可以使用的映射站台，如果不想使用，可以注解到这行；

baseurl=：这个最重要，因为后面接的就是软体库的实际网址！mirrorlist 是由yum 程式自行去捉映射站台， baseurl 则是指定固定的一个软体库网址！我们刚刚找到的网址放到这里来啦！

enable=1：就是让这个软体库被启动。如果不想启动可以使用enable=0 喔！

gpgcheck=1：还记得RPM 的数位签章吗？这就是指定是否需要查阅RPM 档案内的数位签章！

gpgkey=：就是数位签章的公钥档所在位置！使用预设值即可

范例一：列出目前yum server所使用的软体库有哪些？

[root@study ~]# yum repolist all

修改软件源可能会产生的问题与解决

清除掉本机的列表旧数据即可

[root@study ~]# yum clean [packages|headers|all]

选项与参数：

packages：将已下载的软体档案删除

headers ：将下载的软体档头删除

all ：将所有软体库资料都删除！

范例一：删除已下载过的所有软体库的相关资料(含软体本身与清单)

[root@study ~]# yum clean all

YUM 软件群体组功能

[root@study ~]# yum [群组功能] [软体群组]

选项与参数：

grouplist ：列出所有可使用的『软体群组组』，例如Development Tools 之类的；

groupinfo ：后面接group\_name，则可了解该group 内含的所有软体名；

groupinstall：这个好用！可以安装一整组的软体群组，相当的不错用！

groupremove ：移除某个软体群组；

范例一：查阅目前软体库与本机上面的可用与安装过的软体群组有哪些？

[root@study ~]# yum grouplist

Installed environment groups: #已经安装的系统环境软体群组

Development and Creative Workstation

Available environment groups:

[root@study ~]# yum groupinstall "Scientific Support"

正常情况下系统是会帮你安装好各项软体的。只是伤脑筋的是，刚刚好Scientific Support 里面的软体都是『可选择的』！而不是『主要的(mandatory)

如果想要让groupinstall 预设安装好所有的optional 软体呢？那就得要修改设定档！更改选groupinstall 选择的软体项目即可！如下所示：

[root@study ~]# vim /etc/yum.conf

.....(前面省略).....

distroverpkg=centos-release #找到这一行，底下新增一行！

group\_package\_types=default, mandatory, optional

.....(底下省略).....

[root@study ~]# yum groupinstall "Scientific Support"

EPEL/EPRepo 外挂软件以及自定义配置文件

有些软件并不是在centos 7的软件源中,所以就有EPEL 扩展软件源

透过上述的CentOS 7 的EPEL 计画来安装netcdf 以及R 这两套软体，该如何处理？

答：

首先，你的系统应该要针对epel 进行yum 的设定档处理，处理方式如下：

[root@study ~]# vim /etc/yum.repos.d/epel.repo

[epel]

name = epel packages

baseurl = https://dl.fedoraproject.org/pub/epel/7/x86\_64/

gpgcheck = 0

enabled = 0

鸟哥故意不要启动这个软体仓库，只是未来有需要的时候才进行安装，预设不要去找这个软体库！

接下来使用这个软体库来进行安装netcdf 与R 的行为喔！

[root@study ~]# yum --enablerepo=epel install netcdf R

使用本机的的安装光盘安装软件

假设光盘挂载到/mnt

[root@study ~]# vim /etc/yum.repos.d/cdrom.repo

[mycdrom]

name = mycdrom

baseurl = file:///mnt

gpgcheck = 0

enabled = 0

[root@study ~]# yum --enablerepo=mycdrom install software\_name

全系统自动升级

[root@study ~]# echo '10 1 \* \* \* root /usr/bin/yum -y --enablerepo=epel update' > /etc/cron.d/yumupdate

[root@study ~]# vim /etc/ crontab

最好重新启动crond服务,此外还要分析日志文件与邮件,如果有内核相关的升级,那就得要重启机子才能让安装的软件顺利运行

RPM 与 TARBALL 的选择

优先选择原厂的RPM

管理安装方便,还有查询验证数字签名等特性

特殊软件没有RPM就使用TARBALL

用TARBALL测试新版本

新版本可能会不稳定有问题,但又想使用新版本(某些功能),可以用TARBALL安装到/usr/local下面,与旧版的软件同时存在而互不干扰

SRPM的使用

一般来说,需要用到SRPM的文件,大部分原因就是你需要重新修改里面的某些设置,让软件加入某些功能

具体参考鸟哥Linux

X windows

略过,具体参考鸟哥

linux内核编译与设置

由于现在的系统强调在线升级功能,因此不建议自定义内核,此章比较进阶,主要与系统移植有关(系统装到U盘,让Linux驱动小家电),略过